Doctoral Thesis

Design of Efficient Symmetric-Key Cryptographic Algorithms

by
Kosei Sakamoto

March 2023

Graduate School of Applied Informatics
University of Hyogo

Abstract

Cryptography plays a crucial role in the modern internet communication system. Especially, thanks to the development of communication technology, we can see variety of its applications everywhere, and the importance of cryptography is getting more and more enhanced. The main role of cryptography, especially symmetric-key cryptography which is treated in this thesis, is to provide data confidentiality and integrity by a block cipher, stream cipher, hash function, message authenticated codes and authenticated encryption scheme.

With the rapid development of communication technology for a latest few decades, such symmetric-key cryptographic algorithms need to meet not only the security requirement but also implementation requirement on resource-constrained devices such as RFIDs and medical devices. Studying on such resource-constrained algorithms is called *lightweight cryptography*, and have get the most attention in the field of symmetric-key cryptography for the latest decade. Another area of interest is designing ultra-high throughput cryptographic algorithms for the rapid advancement of mobile communication systems like 5G and beyond 5G. In these systems, it is necessary to design ultra-high throughput and high-security cryptographic algorithms due to the increasing of the data transmission speed.

This thesis is dedicated to the design of symmetric-key cryptographic algorithms, including a lightweight block cipher, lightweight tweakable block cipher, block cipher-based low-katency pseudo-random function, and ultra-high throughput uthenticated encryption with associated data scheme. Specifically, we introduce four algorithms: a tweakable block cipher Tweakable TWINE, lightweight block cipher WARP, lightweight PRP Orthros, and an ultra-high throughput AEAD Rocca and investigate how to design them along with the background of why these new algorithms is necessary.

Table of Contents

1 Introduction **1**

1.1 Motivation and Background 1

1.1.1 Applications and Requirements of Efficient and Lightweight Cryptographic Algorithms 2

1.2 Our Contribution and Outline 3

2 Symmetric-Key Cryptography **6**

2.1 Block Cipher 6

2.1.1 Modern Block Cipher Construction 7

2.1.2 Tweakable Block Cipher 11

2.2 Authenticated Encryption 12

2.2.1 Construction of a Sponge-Based AEAD 13

2.3 Cryptanalysis of Symmetric-Key Cryptographic Algorithms 14

2.3.1 Attack Models 15

2.3.2 Goal of Attack 16

2.3.3 Cryptanalysis Techniques 18

3 MILP-Aided Security Evaluation **26**

3.1 Security Evaluation of Differential/Linear Cryptanalysis 27

3.1.1 Constraints 27

3.1.2 Objective Function 31

3.2 Security Evaluation of Impossible differential Cryptanalysis 31

3.3 Security Evaluation of Integral Cryptanalysis 32

3.3.1 Constraints 33

3.3.2 Objective Function 34

4 Tweakable TWINE: Building a Tweakable Block Cipher on Generalized Feistel Structure **35**

4.1 Introduction 35

4.1.1 Organization of this section 37

4.2 Specification 37

4.2.1 Notation 37

4.2.2 Data Processing Part 38

4.2.3 Tweak Scheduling Function 39
4.2.4 Key Scheduling Function 40
4.2.5 Test Vectors 40
4.3 Design Rational 42
4.3.1 Design Goals 42
4.3.2 How to Design Permutation-Based Tweak Scheduling Function 43
4.3.3 Reducing Candidates 44
4.4 Security Evaluation 46
4.4.1 Differential/Linear Attack 46
4.4.2 Impossible Differential Attack 47
4.4.3 Integral Attack 48
4.5 Hardware Implementation Results 48
4.6 Conclusion 49

5 WARP : Revisiting GFN for Lightweight 128-Bit Block Cipher 51
5.1 Introduction 51
5.2 Specification 54
5.3 Design Rationale 56
5.3.1 Branch Size and Permutation 56
5.3.2 S-box 57
5.3.3 Key Schedule 57
5.3.4 Round Constants 58
5.4 Security Evaluation 58
5.4.1 Differential/Linear Attack 58
5.4.2 Impossible Differential Attack 59
5.4.3 Integral Attack 59
5.4.4 Meet-in-the Middle Attack 60
5.4.5 Invariant Subspace Attack 60
5.5 Hardware Performance 61
5.5.1 Nibble Serial Architecture 61
5.5.2 Performance Results 63
5.5.3 Round Based and Round Unrolled Designs 63
5.5.4 More Details about Hardware Implementations 64
5.6 Software Performance 68

5.6.1 On 8-bit AVR Microcontrollers . . . 68
5.7 On High-end Processors . . . 68
5.7.1 More Details of Software Implementations . . . 69
5.8 Conclusion . . . 71

6 Orthros: A Low-Latency PRF 74
6.1 Introduction . . . 74
6.1.1 Low-Latency Encryption . . . 74
6.1.2 Our Design . . . 75
6.2 Specification . . . 76
6.2.1 Key Scheduling Function . . . 76
6.2.2 Round Function of Branch1 and Branch2 . . . 77
6.2.3 Test Vectors . . . 80
6.3 Design Rationale . . . 80
6.3.1 General Construction . . . 80
6.3.2 Toy Ciphers . . . 84
6.3.3 Linear Layer . . . 87
6.3.4 S-box . . . 92
6.3.5 Key Scheduling Function . . . 93
6.4 Security Evaluation . . . 93
6.4.1 Differential/Linear Attack . . . 93
6.4.2 Impossible Differential Attack . . . 94
6.4.3 Integral Attack . . . 97
6.4.4 Invariant Subspace Attack . . . 100
6.4.5 Meet-in-the-Middle Attack . . . 100
6.4.6 Yoyo and Mixture-Differential Attacks . . . 101
6.4.7 Difficulty of Key-Recovery Attacks . . . 101
6.5 Hardware Evaluation . . . 102
6.6 Conclusions . . . 108

7 Rocca: An Efficient AES-Based Encryption Scheme for Beyond 5G 109
7.1 Introduction . . . 109
7.1.1 Background . . . 109
7.1.2 Our Design . . . 110
7.1.3 Organization . . . 111

7.2 Preliminaries . . . 111
7.2.1 Notations . . . 111
7.2.2 The Round Update Function . . . 112
7.2.3 Specification of Rocca . . . 113
7.3 Design Rationale . . . 116
7.3.1 General Construction . . . 116
7.3.2 Criteria for Performance and Security . . . 119
7.3.3 Finding Efficient Structures . . . 121
7.3.4 Loading the Nonce and Key . . . 122
7.3.5 Generating the Ciphertext Blocks . . . 123
7.4 Security Evaluation . . . 124
7.4.1 Differential Attack . . . 124
7.4.2 Forgery Attack . . . 124
7.4.3 Integral Attack . . . 125
7.4.4 State-recovery Attack . . . 126
7.4.5 The Linear Bias . . . 127
7.4.6 The State-recovery Attack Using the Decryption Oracle . . . 128
7.4.7 Other Attacks . . . 128
7.4.8 No Claims . . . 128
7.5 Software Implementation . . . 128
7.5.1 Software Implementation Results on Other CPUs . . . 129
7.6 Conclusions . . . 132

8 Conclusions **133**

Acknowledgements **134**

References **135**

List of Figures

2.1 Overview of a block cipher. 7

2.2 Overview of an iterated block cipher. R denotes one round of the round function. 8

2.3 Overview of Feistel network. 10

2.4 Construction of type-2 generalized Feistel network. 10

2.5 Overview of a tweakable block cipher. 11

2.6 Overview of an AEAD. 13

2.7 Duplex sponge encryption. 14

2.8 Sponge-based stream cipher. 14

2.9 Sponge-based AEAD. 14

2.10 Entire encryption/decryption process of AEGIS, Tiaoxin, and Rocca. 15

2.11 $DR_2 = 5$ on type-2 GFN. Red lines mean to be affected by the input difference of x_2. 22

4.1 Overview of Tweakable TWINE 37

4.2 Encryption and decryption of Tweakable TWINE 38

4.3 S-box S and nibble shuffle π 39

4.4 Tweak scheduling function of Tweakable TWINE 39

4.5 Key schedules of Tweakable TWINE, for 80-bit and 128-bit keys. S-box S is the same as Fig. 4.2, and key schedule constants, CON^i, are described in the bottom. 41

4.6 Permutaion-based tweak scheduling function 43

4.7 Permutation of $d = 4$ 44

4.8 Positions for tweak inputs 44

4.9 18-round impossible differential characteristic 47

4.10 18-round integral characteristic 48

5.1 Round Function of WARP. 54

5.2 Encryption algorithm of WARP. 55

5.3 General LBlock-like round function. 58

5.4 Equivalent round function of WARP in LBlock-like structure. 58

5.5 Alternative definition of Round Function 61

5.6 Nibble serial architecture for WARP. The filter that feeds the permuted roundkey is omitted in the diagram. 62

5.7 Breakdown of component-wise area figures for 3 versions of WARP. Nibble and Bit-serial circuits require lesser scan flip-flops which require more area . 65
5.8 Nibble to bit serial transformations 65
5.9 Sketch of the 3 share nibble serial architecture for WARP 67
5.10 Software performance of WARP, SIMON and SKINNY on the same processor. 71
5.12 Another equivalent form using four permutations for round functions and eight permutations for round keys (omitted addition round constants) 73
6.1 Overview of Orthros. 77
6.2 Algorithms of KSF1 and KSF2. 78
6.3 The round function of Branch1 and Branch2 in the first 4 rounds. 79
6.4 The round function of Branch1 and Branch2 in the last 8 rounds. The nibble permutation and the matrix multiplication in the last round will be omitted. 80
6.5 Algorithms of Branch1 and Branch2, where $N \in \{1, 2\}$. 81
6.6 Processing algorithm of Orthros. 82
6.7 (Left) The toy cipher using a single branch and (Right) that using double branches, where WK and RK are the whitening key and round key, respectively. 84
6.8 4-GFS. Dotted lines denote (random) round keys. 86
6.9 Transition of a state after applying P_{bk1}. 89
6.10 The 4 active bits after the bit permutation in the first round, as marked in red. 91
6.11 The 12 active nibbles after MatrixMul in the first round, marked in red. . . . 91
6.12 The 48 active bits after bit permutation in the second round, marked in red. 91
6.13 The active nibbles after bit permutation in the second round, marked in red. 91
7.1 Illustration of the round function 112
7.2 The procedure of Rocca 114
7.3 The process of aesenc for Intel Ice-lake. 117
7.4 The general construction considered of the round function in [103]. Dash lines mean that it can be possible to be absent or present in the design. . . . 118
7.5 General construction of the round function. Dash lines mean that it can be possible to be absent or present in the design. 118
7.6 The round function whose # of blocks is 8. 119
7.7 The round function whose # of blocks is 9. 120
7.8 The round function whose # of blocks is 10. 120

List of Tables

4.1 Permutation in the tweak scheduling function π^t 40

4.2 The number of active S-box of each round for 32 tweak scheduling functions with $d = 6$ that achieve 32 active Sbox in 19 rounds. 45

4.3 Lower bound on the number of differentially and linearly active S-boxes in each model 47

4.4 Hardware results for round-based, enc-only implementations. 49

4.5 Hardware results for multi-round, enc-only implementation of T-TWINE-80. (Top) ASIC (Yosys with `osu018_stdcells.lib`) (Bottom) FPGA (Intel/Altera `10CL120YF780I7G`, Quartus 18.1) 50

5.1 4-bit S-box S. 54

5.2 Shuffle π on 32 nibbles. 54

5.3 Round constants (listed in hexadecimal). 55

5.4 Test vectors. 56

5.5 Four equivalent classes of 32-branch permutations with 9-round full diffusion [73]. 57

5.6 Lower bounds on the number of Active S-boxes for WARP and four permutations of $\pi'_0(x)$, $\pi'_1(x)$, $\pi'_2(x)$, and $\pi'_3(x)$ 57

5.7 The lower bound for the number of differentially and linearly active S-boxes in the single-key setting. 59

5.8 Comparison of performance metrics for serial implementations synthesized with STM 90nm Standard cell library. Figures separated by / indicate corresponding metrics for encryption/decryption. *Synthesized with the IBM 130 nm process/Power at 100 KHz 63

5.9 Comparison of performance metrics for round based implementations synthesized with STM 90nm Standard cell library (1R, 2R, 4R refer to 1, 2, and 4 round unrolled circuits). 64

5.10 Comparison of performance metrics for serial implementations synthesized with STM 90nm Standard cell library. (RB denotes round based circuit, 3s, 4s denotes circuits with 3, 4 shares respectively) *Synthesized with the UMC 180nm process/Power at 100 KHz. **Synthesized with the IBM 130nm process/Power at 100 KHz 67

5.11 Different performance characteristics of WARP on 8-bit AVR 69

5.12 Performance of block ciphers (128-bit block and 128-bit key) on 8-bit AVR . 70

5.13 Software performance profile of WARP with various message length (including the time took by packing/unpacking messages) 71
6.1 S-box in Branch1 and Branch2. 77
6.2 Specification of the round constants RC_r^1 and RC_r^2. 79
6.3 Bit permutation P_{bkN} for key scheduling KSFN, where $N \in \{1, 2\}$. 82
6.4 Bit permutation P_{brN} for round function Branch$N$, where $N \in \{1, 2\}$. 83
6.5 Nibble permutation P_{nN} for round function Branch$N$, where $N \in \{1, 2\}$. . . 83
6.6 Test vectors for Orthros in hex. 84
6.7 The maximal differential probability of SPN-based toy ciphers. 85
6.8 The maximal differential probability for each GFS-based construction. 86
6.9 The upper bound of the number of active bits after each operation. 88
6.10 Comparison of lower bounds of the number of active S-boxes. 92
6.11 Comparison of S-boxes. 93
6.12 The lower bounds of the number of active S-boxes in the single-key setting. 94
6.13 The differential distribution table of S-box. 95
6.14 The propagation of the division property for the S-box. 98
6.15 The mapping of the binary matrix. 98
6.16 The propagation of the division property for the binary matrix. 99
6.17 Results for the STM 90nm library. Power measured at 10 MHz. *The core implementation of the underlying permutations in these constructions were taken from [35, 64] 104
6.18 Results for the Nangate 15nm library. Power measured at 10 MHz. 105
6.19 Results for the TSMC 90nm library. Power measured at 10 MHz. 106
6.20 Results for the Nangate 45nm library. Power measured at 10 MHz. 107
7.1 Test vectors. 116
7.2 Latency and throughput of aesenc for some architectures by Intel and AMD referred by [148]. 117
7.3 Comparison of the performance of the round function having different number of blocks at the same *rate*. 120
7.4 Round functions of AEGIS family and Tiaoxin-346 121
7.5 Candidates of round functions which we search. 122
7.6 Speed (in cycles / Byte) of round functions of Rocca, AEGIS-128, AEGIS-128L, AEGIS-256, Tiaxion-346, and JN16 (not include a generation part of a ciphertext). 122

7.7 The lower bound for the number of active S-boxes in the initialization phase where AS_{sk} and AS_{rk} mean an active S-box in the single-key setting and in the related-key setting, respectively. 124
7.8 Performance Evaluation 129
7.9 Performance on Intel(R) Core(TM) i9-12900K CPU with 64 GB RAMs. . . . 130
7.10 Performance on Intel(R) Core(TM) i9-11900 CPU@2.50GHz with 64 GB RAMs. 130
7.11 Performance on Intel(R) Core(TM) i9-10910 CPU@3.60GHz with 64 GB RAMs. 130
7.12 Performance on Apple M1 131
7.13 Performance on Apple A15 Bionic 131
7.14 Performance on Qualcomm Snapdragon 888 131

1 Introduction

Cryptography is a mathematical science aiming at realizing secure communication systems. Its study has a long history, dating back over 4000 years, and has resulted in the development of numerous cryptographic algorithms. These algorithms can be divided into modern ciphers and other types. The distinguishing characteristic of modern ciphers is that their encryption and decryption algorithms are made public, while the algorithms used by other types of ciphers are not. This criterion was used in the selection of the *Data Encryption Standard* (DES), a former American standard block cipher, and remains a fundamental requirement for modern cipher design. The study of modern cryptography has undergone significant advancement, thanks to the tireless efforts of numerous cryptographers.

Cryptography has played a vital role in modern internet communication society, and we can see its technology in a wide range of devices and systems, including smartphones, personal computers, and cloud servers to name a few. Thus, studying modern cryptography is essential to maintain and progress the security of our internet society. The primary role of modern cryptographic technology is to provide not only *confidentiality* but also *integrity* and *authenticity*.

Confidentiality An unauthorized third party must not be able to obtain any knowledge of the original content of the communication by intercepting the messages during transmission. In addition, stored data must be protected from unauthorized access.

Integrity The recipient must be able to verify that the content of the transmitted information was not modified during transmission.

Authenticity The recipient must be able to verify the origin of the message, ensuring that it was indeed sent by the expected sender.

In general, we can roughly classify the study of cryptography into design and cryptanalysis. In the field of symmetric-key cryptography, researchers typically approach it from both design and cryptanalysis perspectives and strive to advance both areas by sharing insights with one another. In this thesis, we primarily focus on the design of symmetric-key cryptography.

1.1 Motivation and Background

The rapid development of communication technology has made cryptography increasingly important in a variety of fields, including automotive communication, RFID, and industrial control networks to name a few. This rapid development brought about some new interests in the field of symmetric-key cryptography.

Over the past decade, a major area of focus in the field of symmetric-key cryptography has been the design of efficient cryptographic algorithms that meet certain constraints, such as circuit area, memory, energy, and latency. This research area is known as ”lightweight cryptography,” and ”lightweight cryptographic algorithms” are in high demand across a range of industries, including automotive communication, RFID, and industrial control networks to name a few. In fact, the National Institute of Standards and Technology (NIST) has initiated the standardization of lightweight cryptographic algorithms (LWC) since 2018[1], as a concern that the current standard block cipher AES cannot meet the requirement in some applications has come up. This standardization process is open, meaning that a standard lightweight cryptographic algorithm will be selected from among many candidates submitted by cryptographic researchers worldwide similar to the standardization process of AES and SHA3. As of November 2022, the process has progressed to Round 3 (the final round), and 10 out of 57 submissions have been selected as final-round candidates. Since going forward the limits of lightweight cryptography for both industry and academia is valuable and important, many researchers dedicate significant effort to this area.

Another area of interest in this field is the design of ultra-efficient symmetric-key cryptographic algorithms that take full advantage of the functions provided by modern popular devices, such as SIMD instructions and AES new instruction sets (AES-NI). Fifth-generation mobile communication systems (5G) and beyond systems, called beyond 5G or 6G, can achieve ultra-high-speed data transmission, with the 6G system expected to achieve speeds exceeding 100 Gbps [117]. To prevent cryptographic algorithms from becoming a bottleneck in achieving such ultra-high speeds, it is crucial that the cryptographic algorithms used in these systems also be able to perform encryption and decryption at speeds exceeding 100 Gbps on a pure software environment. However, the current standard symmetric-key cryptographic algorithms for the 4G system only achieve speeds of 30 Gbps [178], falling short of the requirements for beyond 5G systems. Additionally, these systems must also have a key length of at least 256 bits to ensure sufficient security against quantum computers. In general, performance and security are the relation of trade-off in symmetric-key cryptographic algorithms, making it a challenging task for algorithm designers to enhance both simultaneously.

1.1.1 Applications and Requirements of Efficient and Lightweight Cryptographic Algorithms

As mentioned before, with the recent rapid development of communication technology, various applications that require such as ultra-low latency and high throughput are coming up. In addition,

[1] https://csrc.nist.gov/Projects/lightweight-cryptography

the recent IoT devices have strict resource restrictions for hardware area and memory size, and it makes the resource that cryptographic algorithms can utilize really few in those devices. We here give some of their applications and their requirements for symmetric-key cryptographic algorithms as examples. While there are many other applications that have other restrictions about, such as memory and energy, we here focus on applications that have requirements about hardware area, latency, and high throughput all of which are treated in this thesis.

Small hardware area One of the most famous applications that requires a small hardware area is RFID. RFID is primarily used for uniquely identifying an object, animal, or person through wireless communication and is utilized in a variety of settings, such as unmanned cash registers and IC cards. In fact, the designer of the first lightweight block cipher PRESENT mentioned that AES is not suitable for using in RFID tags and sensor networks in their paper [51], and this is one of motivations that they proposed PRESENT. According to some works [156, 172], the cryptographic system in RFID should be realized with a circuit of 3000-4000 Gate Equivalents (GEs) or less.

Low latency There are many applications requiring a quick response, such as encryption of memory bus, storage systems, and automotive communication. In particular, it is said that several technologies realized in beyond 5G system, such as edge computing, require a cryptographic system with an encryption/decryption process of less than 1ns at most. As the current de facto standard block cipher AES is hard to achieve its latency of less than 1.5 ns [8], the new symmetric-key cryptographic algorithm is essential.

High throughput The upcoming beyond 5G system is expected to achieve ultra-high data transmission speeds for a range of applications. For instance, multi-sensory XR requires 20 times the data transmission speed of the 5G system (100 - 400 Gbps), and real-time remote applications such as telesurgery and ultra-high quality holograms require data transmission speeds of 0.5 - 1 Tbps. As the current standard symmetric-key cryptographic algorithms for the 5G system only achieve around 50 Gbps of encryption/decryption speed, it is clear that a new ultra-high throughput cryptographic algorithm is necessary.

1.2 Our Contribution and Outline

In this thesis, we present strategies for designing efficient symmetric-key cryptographic algorithms. Our focus is on the design of block ciphers, a dedicated AEAD, and the introduction of strategies for designing efficient symmetric-key algorithms. Before providing the main parts, we provide definitions, security requirements, and cryptanalysis techniques for symmetric-key

cryptographic algorithms. Additionally, we present an automatic security evaluation method with Mixed Integer Linear Programming (MILP). The main part of this thesis consists of four sections: design of a tweakable lightweight block cipher Tweakable TWINE [155], lightweight block cipher WARP [10], block cipher-based low-latency pseudo-random function Orthros [18], and ultra-high throughput encryption with associated data (AEAD) scheme Rocca [154]. The details of these main sections are as follows:

Section 4 We first introduce the lightweight tweakable block cipher Tweakable TWINE, which aims to minimize hardware circuit area. A tweakable block cipher is a variant of a block cipher, a symmetric-key cryptographic algorithm, that has an additional public input called a "tweak." The key feature and difference of a tweakable block cipher from an ordinary block cipher is that the attacker can know and control the tweak during the attack process, as the tweak is a public input. As a result, designers of a tweakable block cipher must take this attack scenario into consideration. Generally, this extra input weakens the cipher and presents a greater challenge for the designers to evaluate its security. In Sect. 4, we demonstrate how to build a new tweakable block cipher on an existing block cipher with the proposal of Tweakable TWINE. As an initial security evaluation for Tweakable TWINE, we evaluate its security against differential, linear, integral, and impossible differential attacks, which are the prime attacks against symmetric-key cryptographic algorithms and are commonly used as initial security evaluations when designing a new cipher. In terms of hardware implementation results, Tweakable TWINE performs competitively in terms of hardware circuit area, as measured by gate equivalent (GE), compared to other tweakable and ordinary block ciphers.

Section 5 As another lightweight block cipher targeting low hardware implementation area, we introduce a 128-bit lightweight block cipher WARP. WARP is based on the Feistel structure, one of the prime constructions for block ciphers. The main challenge in designing WARP is selecting a good permutation in a linear layer from numerous candidates. Since WARP is designed as a nibble operation-oriented cipher, with all operations following a nibble-wise operation, we must choose a good permutation from $(16!)^2$ candidates, which is discussed more detailly in Sect 5. To address this challenge, we convert the general construction of WARP into a Lblock-like construction that is equivalent in terms of security. This conversion significantly reduces the number of permutation candidates from $(16!)^2$ to $8!$, allowing us to find the best one among them. The initial security evaluation for WARP is similar to that of Tweakable TWINE, with the addition of evaluations against invariant and meet-in-the-

middle attacks. In terms of hardware implementation results, WARP achieves the smallest hardware circuit area among all 128-bit block ciphers, with a hardware circuit area of fewer than 1000 GEs for the first time.

Section 6 In addition to designing a cipher with a low hardware implementation area, it is also interesting to explore how low-latency a cipher can be developed. To this end, we introduce a low-latency pseudo-random function (PRF) called Orthros. A distinctive feature of Orthros is that it does not have a decryption procedure, which this type of cipher is referred to as PRF. This feature is derived from Orthros being based on a two-branch construction, in which the branches are relatively weak pseudo-random permutations (PRPs) and the output is calculated by XORing the outputs of the two branches. With this two-branch construction, Orthros achieves the lowest latency among other ciphers. A major challenge in designing Orthros is the selection of a good permutation, as well as in WARP. The primary difference between the two is the number of candidate permutations and the approach used to find a suitable one. For Orthros, which is a bit-oriented cipher with a 128-bit block, the number of candidate permutations is $128!$. To reduce the search space, we employ a two-step approach. We provide the detailed method for finding a good permutation in Orthros-type ciphers through an explanation of Orthros.

Section 7 Lastly, we present Rocca, an ultra high-throughput authenticated encryption with associated data (AEAD) scheme. Rocca is designed for beyond 5G applications and includes a 256-bit key to protect against attacks by quantum computers. Unlike the previously mentioned block ciphers, which only provide confidentiality, Rocca is an AEAD that ensures both confidentiality and integrity. The main feature of Rocca is the use of a sponge-based round function to achieve ultra-high throughput. Additionally, Rocca takes full advantage of SIMD instructions and AES-NI, which allow for efficient execution of basic operations like XOR and multiplication, as well as an AES round function, on a pure software environment. As a result, Rocca achieves over 200 Gbps on the latest CPU, meeting the requirements for beyond 5G in terms of both performance and security, throughput of over 100 Gbps and support for a 256-bit key. Note that Rocca is the first dedicated symmetric-key cryptographic algorithm meeting these requirements for beyond 5G.

2 Symmetric-Key Cryptography

Symmetric-key cryptography plays a crucial role in contemporary internet communication systems. One of the defining features of symmetric-key cryptographic algorithms is their use of identical secret keys for both encryption and decryption processes, meaning that two parties must previously share a secret key in order to securely communicate. Contrastingly, public-key cryptography (or asymmetric cryptography) does not require the pre-sharing of secret information. In symmetric-key cryptography, the unencrypted (original) information is referred to as plaintext, and the encrypted information that is transmitted through public channels such as the internet is referred to as ciphertext. If a plaintext M is encrypted using a secret key K, the resulting ciphertext C can be decrypted back into the original plaintext P using the same secret key K, as follows:

$$C = \text{Enc}(K, M), \quad M = \text{Dec}(K, C),$$

where Enc() and Dec() denote the encryption and decryption algorithms, respectively.

Generally, symmetric-key cryptography encompasses a range of algorithms including block ciphers, stream ciphers, hash functions, message authenticated codes (MAC), and authenticated encryption with associated data (AEAD). Among them, we deal with block ciphers and AEADs in this thesis.

2.1 Block Cipher

A block cipher is utilized to ensure data confidentiality. As can be seen in the name of "block cipher", a block cipher converts a plaintext of a certain length, referred to as a block, typically 64 or 128 bits in many block ciphers, into a ciphertext of the same length using a secret key. Fig 2.1 illustrates an overview of a block cipher. More formally, it can be defined as follows:

Definition 1 Block cipher: *An n-bit block cipher E_K with a k-bit secret key $K \in \mathbb{F}_2^k$ is a keyed function:*

$$E_K : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n,$$
$$\{M, K\} \to C,$$

where $M \in \mathbb{F}_2^n$, $C \in \mathbb{F}_2^n$ denote a plaintext and ciphertext, respectively. The inverse function of E_K is expressed as E_K^{-1} if a block cipher is inversible.

Most block ciphers have a corresponding decryption procedure, making them bijective functions. Since these block ciphers can be viewed as deterministic permutations that depend on a

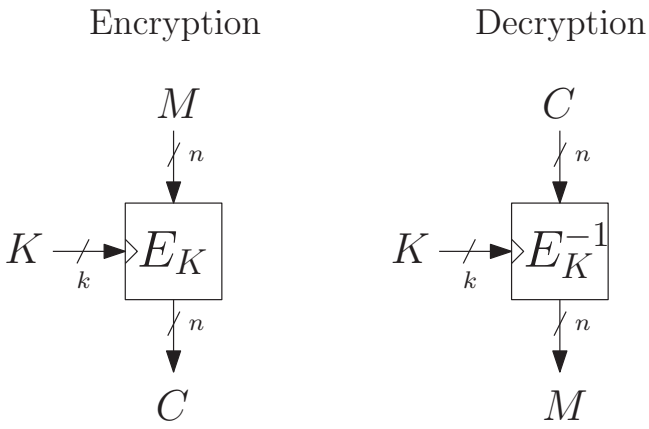


Figure 2.1: Overview of a block cipher.

secret key, they are known as pseudo-random permutations (PRP). On the other hand, some block ciphers do not have a decryption procedure and are therefore not bijective functions. In this case, these ciphers are referred to as pseudo-random functions (PRF).

2.1.1 Modern Block Cipher Construction

The design concept of symmetric-key cryptographic algorithms is grounded in the principles of confusion and diffusion, as introduced by Shannon and abstracted by Massey [122, 159]:

Confusion The statistical properties of the ciphertext should depend on the statistical properties of the plaintext in a way that is too complex for an attacker to distinguish.

Diffusion Each bit of the plaintext and each bit of the secret key should influence as many bits of the ciphertext as possible.

Generally, a block cipher can be divided into a data processing part and a key scheduling part. The key scheduling part expands a given secret key, also known as a master key, into several round keys which are inserted one by one throughout the cipher. The data processing part uses the round keys generated by the key scheduling part to encrypt a given plaintext and produce a ciphertext. Almost all modern block ciphers have an iterated construction, meaning that the data processing part is composed of a simple function called a round function, which is called multiple times in the data processing part.

Definition 2 Iterated block cipher: *A block cipher constructed as an iterated construction is referred to as an iterated block cipher. The data processing part of an iterated block cipher* E_d

can be expressed as follows:

$$E_d(\cdot) = f_r(\cdot) \circ \cdots \circ f_2(\cdot) \circ f_1(\cdot),$$

where f_i and rk_i denote the i-th round function and the round keys.

A round function usually consists of very simple mathematical operations, such as an XOR, AND, modular addition, and look-up table called *S-box*. Fig. 2.2 illustrates the overview of an iterated block cipher.

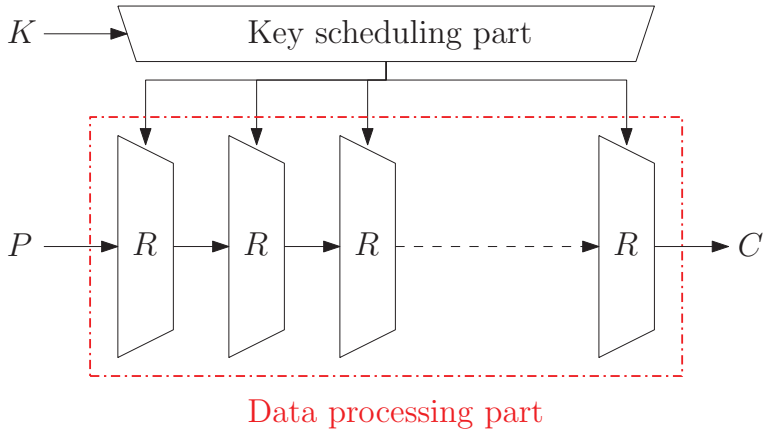


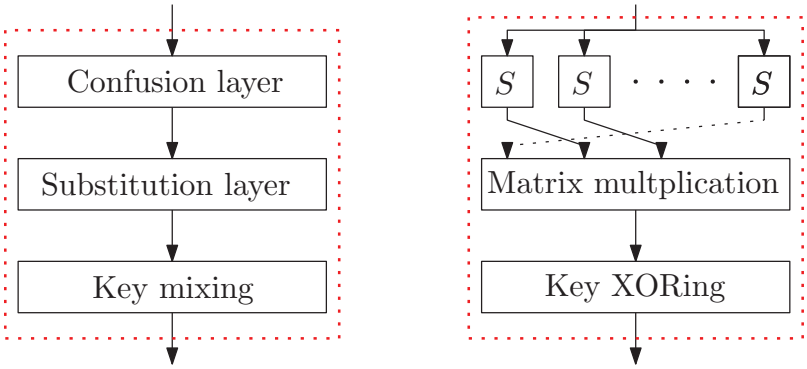
Figure 2.2: Overview of an iterated block cipher. R denotes one round of the round function.

It should be mentioned that some block ciphers do not consist of a single round function in their data processing part, but rather a few similar round functions. Strictly speaking, these ciphers are not iterated ciphers, but in the context of symmetric-key cryptography, they can be also considered as iterated block ciphers.

As popular structures of a block cipher, there are three ones called *Substitution Permutation Network* (SPN), *Feistel network*, and *And rotation XOR* (ARX). The SPN and Feistel structures will be discussed in the next section.

2.1.1.1 Substitution Permutation Network The fundamental idea behind this type of construction can be traced back to Shannon's cipher design principle [159], in which a block cipher is constructed by substitution and diffusion layers. With this idea, the round function in SPN-based block ciphers consists of a key mixing, diffusion layer, and substitution layer.

Definition 3 Substitution permutation network (SPN): *A substitution permutation network is a structure for block ciphers that is based on a series of key mixing layers, substitution layers, and diffusion layers.*



(a) Overview of an SPN-based block cipher

(b) The detailed construction of an SPN-based block cipher

Key mixing is often realized through an XOR operation in modern ciphers. The diffusion layer is implemented through simple linear operations such as XOR, permutation, and matrix multiplication. The substitution layer is typically implemented using a substitution box (S-box), which is a look-up table. Many modern SPN-based block ciphers apply these operations on a word-by-word basis (nibble-wise, byte-wise), and these operations are typically performed in parallel. Fig.2.3a and 2.3b provide an overview and detailed depiction of SPN-based ciphers.

Many SPN-based block ciphers follow this design strategy. One of the most notable examples is the Advanced Encryption Standard (AES) [67], which is the standard block cipher algorithm of the United States. In this thesis, we introduce a low-latency pseudo-random function (PRF) Orthros [18], as an SPN-based block cipher.

2.1.1.2 Feistel Network In Feistel network-based block ciphers (referred to as Feistel-based block ciphers in this thesis), the basic operations applied are the same as those in an SPN-based block cipher, such as XOR, permutation, matrix multiplication, and S-boxes. The main difference from SPN is the way these basic operations are applied to the plaintext (internal state). In the case of a Feistel network, the plaintext is first divided into two internal states, and these states are alternately applied to linear and non-linear operations in each round.

Definition 4 Feistel network: *A Feistel network is a structure for block ciphers that divides a plaintext M into two states, denoted $M = L_i||R_i$, where i denotes the round number. The divided states L_i and R_i are alternatively applied to a cryptographic function F_c in each round as follows:*

$$R_{i+1} = L_i,$$

$$L_{i+1} = F_c(R_i, rk_i) \oplus L_i$$

where rk_i denotes the round key in the ith round.

Fig. 2.3 illustrates the overview of Feistel network.

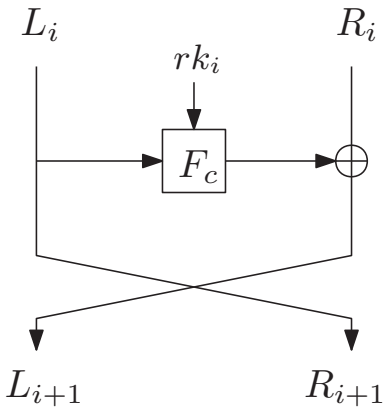


Figure 2.3: Overview of Feistel network.

Like SPN-based block ciphers, most Feistel-based block ciphers also apply basic operations in parallel on a nibble and byte-wise basis. Fig. 2.4 shows the detailed construction of a type-2 generalized Feistel network, a popular variant of the Feistel network, as an example of the parallel application of basic operations.

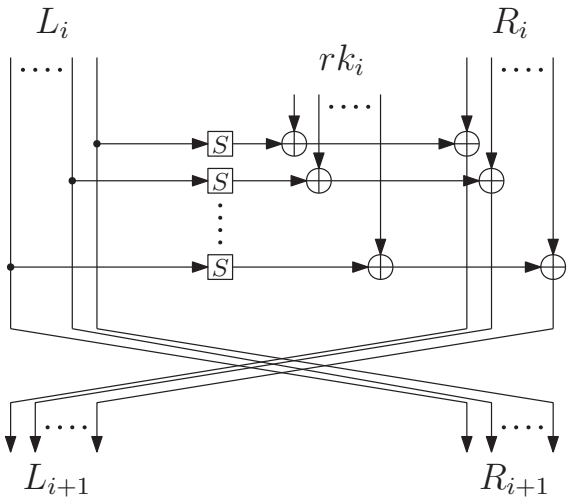


Figure 2.4: Construction of type-2 generalized Feistel network.

It is clear that the number of rounds for Feistel-based block ciphers tends to be larger than that of SPN-based block ciphers, as the internal state in a Feistel network is encrypted half by half in each round. However, Feistel-based block ciphers have merit in terms of hardware implementation that Feistel-based block ciphers only require an encryption circuit for both encryption and decryption procedures in hardware implementation, while SPN-based block ciphers require both encryption and decryption circuits individually.

There are fewer Feistel-based block ciphers compared to SPN-based block ciphers. However, several important block ciphers are based on the Feistel network, such as the Data Encryption Standard (DES) [48], the previous standard block cipher algorithm of the United States. In this thesis, we introduce two Feistel-based block ciphers: Tweakable TWINE [155] and WARP [10]. Note that Tweakable TWINE is not a typical block cipher, but a "tweakable block cipher" that will be described in the next section.

2.1.2 Tweakable Block Cipher

A tweakable block cipher (TBC) is a variant of a block cipher, which has an extra input called *tweak*. The primary purpose of introducing a TBC is to provide beyond-birthday bound security [62]. The formal concept of a TBC was introduced by Rivest and Wagner [118]. We give the definition of a TBC as follows:

Definition 5 Tweakable block cipher: *An n-bit tweakable block cipher E_T with a k-bit secret key $K \in \mathbb{F}_2^k$ and a t-bit tweak $T \in \mathbb{F}_2^t$, which a tweak is a public value, is a cryptographic function as follows:*

$$E_T : \mathbb{F}_2^n \times \mathbb{F}_2^k \times \mathbb{F}_2^t \rightarrow \mathbb{F}_2^n$$
$$\{M, K, T\} \rightarrow C,$$

where $M \in \mathbb{F}_2^n$ and $C \in \mathbb{F}_2^n$ denote a plaintext and ciphertext, respectively. The inverse function of E_T is expressed as E_T^{-1}.

Fig 2.5 illustrates the overview of a tweakable block cipher.

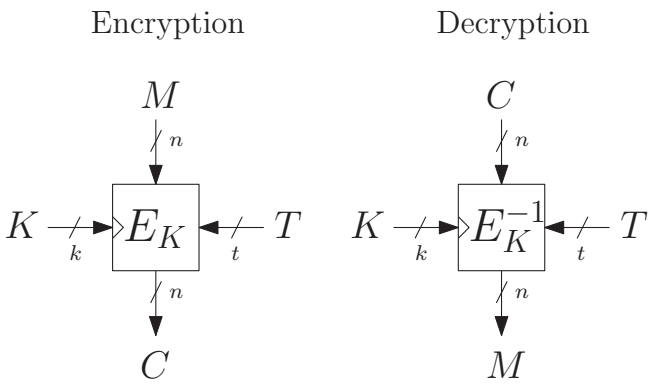


Figure 2.5: Overview of a tweakable block cipher.

One drawback of a TBC is that tweak can be leveraged for cryptoanalysis because a tweak is public. Consequently, the designers of TBCs must consider the possibility of an attacker leveraging a tweak. Despite numerous studies on block cipher-based TBCs [57, 104, 115, 130], there

remains a significant efficiency gap between ordinary block ciphers and TBCs. Nevertheless, TBCs remain popular due to their ability to provide beyond-birthday bound security, and one of the finalists in the NIST LWC competition Romulus [100] employs a tweakable block cipher SKINNY [26] as its underlying primitive. In this thesis, we will introduce a tweakable block cipher Tweakable TWINE [155].

2.2 Authenticated Encryption

Authenticated encryption (AE) scheme ensures both data confidentiality and integrity, while a block cipher only ensures data confidentiality. In general, AE schemes take an unrestricted plaintext, a secret key, and a nonce (a randomly generated value) as inputs, and produce a ciphertext of the same size as the plaintext along with a tag that is used to verify the integrity of the plaintext. The tag is dependent on the plaintext, meaning it is a unique value for a given secret key, plaintext, and nonce.

Authenticated encryption with associated data (AEAD) is a variant of AE that has an additional input called associated data. Associated data is not encrypted, unlike plaintext, but is used in generating the tag, that is, a value of the tag is unique for a plaintext, associated data, secret key, and nonce. This thesis focuses on AEAD, and thus we provide the following definition:

Definition 6 Authenticated encryption with associated data (AEAD): *An authenticated encryption function E_{AE} is a cryptographic function that takes a k-bit secret key K, an n-bit nonce N, a plaintext M with arbitrary size p, and associated data A with arbitrary size a as inputs, and produces a ciphertext C with the same size as the plaintext and a t-bit authentication tag T:*

$$E_{AE} : \mathbb{F}_2^k \times \mathbb{F}_2^n \times \mathbb{F}_2^p \times \mathbb{F}_2^a \rightarrow \mathbb{F}_2^p \times \mathbb{F}_2^t$$
$$\{K, N, M, A\} \rightarrow \{C, T\},$$

Note that associated data A is not encrypted during the encryption process. The inverse function D_{AE} is defined as follows:

$$D_{AE} : \mathbb{F}_2^k \times \mathbb{F}_2^n \times \mathbb{F}_2^p \times \mathbb{F}_2^a \times \mathbb{F}_2^t \rightarrow \mathbb{F}_2^p \cup \{\perp\},$$
$$\{K, N, C, A, T\} \rightarrow M \text{ or } \perp,$$

where $\perp$ denotes an error for all other inputs.

As can be seen in this definition, the integrity of the data can be verified as the owner of the secret key K is the only person who can generate a correct tag T. Fig.2.9 shows an overview of AEAD and secret key confidentiality.

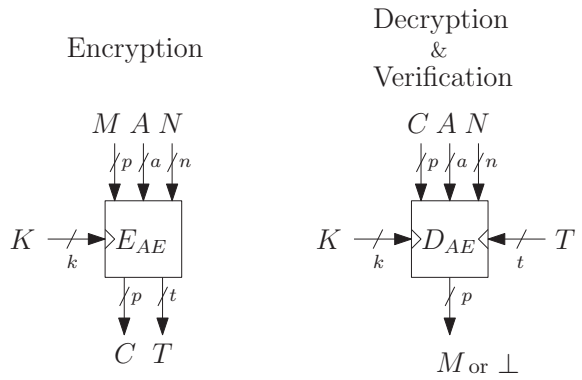


Figure 2.6: Overview of an AEAD.

Currently, there are two popular methods for constructing an AEAD. One approach involves using a block cipher with an additional outer function to provide data confidentiality, i.e., using a block cipher to provide data confidentiality and an additional outer function to provide data integrity. This is the most popular approach since these schemes can claim their security in the context of provable security under a secure block cipher and stream cipher. AES-GCM [1] and ChaCha20-Poly1305 [138] are well-known AEADs followed this approach.

Another approach is to design a dedicated AEAD from scratch. In this approach, it is not possible to claim security in the context of provable security. However, these dedicated AEADs can be much more efficient than the previously mentioned AEADs. In fact, several dedicated AEADs submitted to the CAESAR competition[2] demonstrate impressive throughput for their encryption/decryption process, such as AEGIS, the final portfolio for high-performance applications, and Tiaoxin. This thesis introduces an AEAD called Rocca, which follows this approach. The basic construction of this type of AEAD is explained in the next section.

2.2.1 Construction of a Sponge-Based AEAD

Sponge-based AEADs have a large-state permutation as its core component. The idea behind this construction was introduced by Bertoni et al. [37] as the form of a *duplex sponge encryption* shown in Fig. 2.7. The basic construction of ultra-high throughput dedicated AEADs such as AEGIS, Tiaoxin, and Rocca can be seen as a combination of a duplex sponge encryption and a sponge-based stream cipher introduced by Bertoniet al. [36] as shown in Fig. 2.8. The advantage of this construction is that it allows for simultaneous absorbing of a plaintext and squeezing of ciphertext. It leads to faster encryption/decryption procedures compared to constructions that divide the process into separate absorbing and squeezing phases. We show this sponge-based

[2] `https://competitions.cr.yp.to/caesar.html`

AEAD construction in Fig. 2.9.

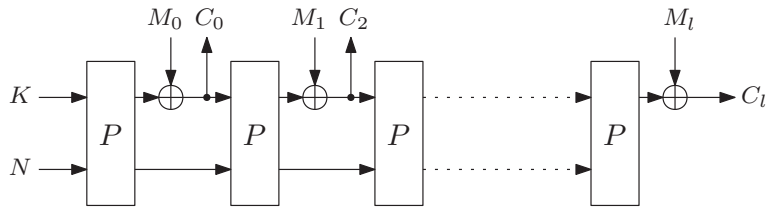


Figure 2.7: Duplex sponge encryption.

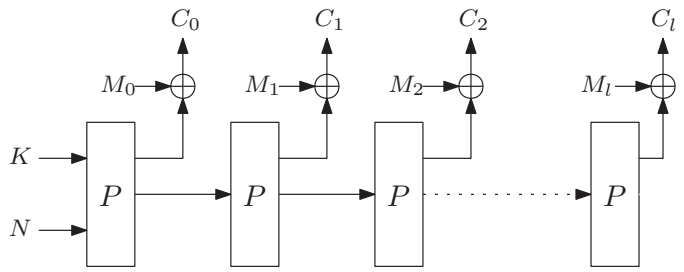


Figure 2.8: Sponge-based stream cipher.

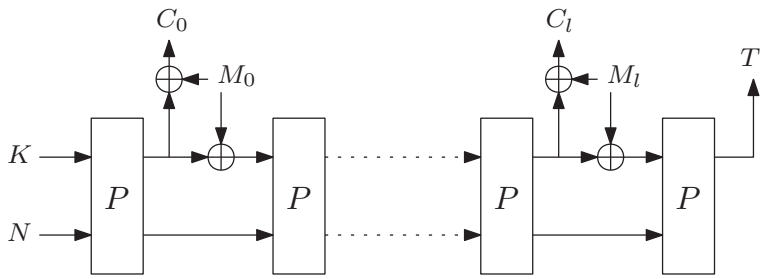


Figure 2.9: Sponge-based AEAD.

The encryption/decryption process of AEGIS, Tiaoxin, and Rocca consists of four phases: initialization, processing the associated data, encryption, and finalization as shown in Fig. 2.10. The core permutation P is initialized in the initialization phase, and the associated data is absorbed in the processing the associated data phase. Ciphertext blocks are then generated by absorbing message blocks, and the tag is generated in the finalization phase.

2.3 Cryptanalysis of Symmetric-Key Cryptographic Algorithms

Cryptanalysis aims at not only breaking symmetric-key cryptographic algorithms, but also evaluating their security. Over the past few decades, various cryptanalysis techniques have been developed and have played a significant role in shaping design strategies.

Cryptanalysis to modern symmetric-key cryptographic algorithms such as block ciphers, stream ciphers, and AEADs can be used to estimate their security margin by evaluating the gap between

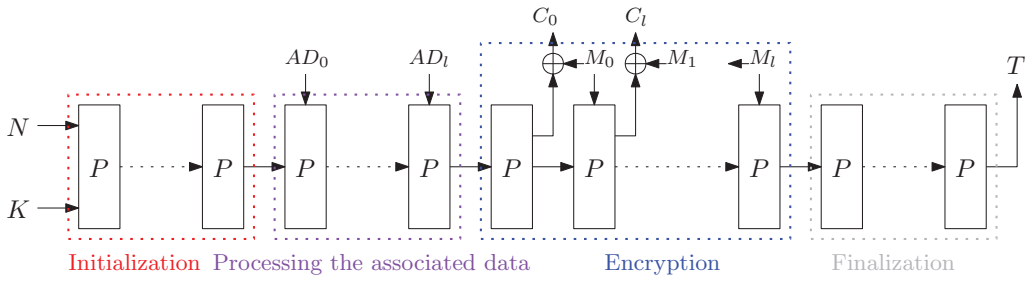


Figure 2.10: Entire encryption/decryption process of AEGIS, Tiaoxin, and Rocca.

the full number of rounds and the number of rounds that can be attacked. For example, the current best attack against AES-128, which has 10 rounds as the full number of rounds, can attack the 7-round reduced AES-128 [142], indicating a security margin of 3 rounds for AES-128. Once an attack is able to reach the full number of rounds, the cipher is considered fully broken. Evaluating the security margin is important for designers in determining security claims for their ciphers and in estimating the current level of security of a cipher.

Currently, there are various cryptanalysis techniques that can be generally classified into generic attacks and dedicated attacks that exploit specific structural properties of algorithms. A generic attack is an inevitable attack for all symmetric-key cryptographic algorithms, such as *brute force attack*, which tries all key candidates and find a correct one. Therefore, the efficiency of dedicated attacks is evaluated by comparing with a generic attack, and the dedicated attack regards as success if the efficiency outperforms that of a generic attack. Therefore, a secure symmetric-key cryptographic algorithm must be resistant to all attack vectors except for generic attacks. In the following sections, we will describe the goals of attacks, attack models, and several popular cryptanalysis techniques.

2.3.1 Attack Models

There are several attack models that are classified based on the level of knowledge and capabilities of the attacker.

Ciphertext-only attack In this type of attack, the attacker only has access to the ciphertext produced by the encryption system. It is assumed that the ciphertext is uniformly random, given that the plaintext is also uniformly random.

Known plaintext attack In this type of attack, the attacker has access to pairs of plaintext and corresponding ciphertext. It is important to note that the attacker does not have control over the plaintext.

Chosen plaintext/ciphertext attack In this type of attack, the attacker has access to an encryption/decryption system and can choose the plaintext/ciphertext to be encrypted/decrypted. This means that the attacker can obtain pairs of arbitrarily chosen plaintext/ciphertext and the corresponding ciphertext/plaintext.

Adaptively chosen plaintext/ciphertext attack This type of attack is similar to the chosen plaintext/ciphertext attack, with the added capability for the attacker to obtain pairs of arbitrarily chosen plaintext/ciphertext and the corresponding ciphertext/plaintext even after the chosen plaintext/ciphertext has been encrypted/decrypted by the system.

In general, the attack models become increasingly advantageous for the attacker as they progress in the above list. In symmetric-key cryptography, symmetric-key cryptographic algorithms must be resistant to all known attack vectors in all of the above attack models.

2.3.2 Goal of Attack

The primary goal of attacks on symmetric-key cryptographic algorithms is to recover the secret key. However, the specific goals of an attack may vary depending on the variant of the symmetric-key cryptographic algorithm. In this section, we will discuss the goals of attacks on block ciphers and AEADs separately.

2.3.2.1 For Block Ciphers According to Kerckhoffs's principle [107], block ciphers should be resistant to any attack vector even if the encryption/decryption algorithm is public. In this context, the goal of attacks on block ciphers are as follows:

Key recovery attack Let k denote the size of the secret key. The attacker aims to recover the secret key from the key space 2^k. Once the secret key has been recovered, the attacker can access the plaintext and internal states of the block cipher.

Distinguishing attack Let k and n denote the size of the secret key and block, respectively. The attacker aims to distinguish the block cipher, which is a set of 2^k n-bit permutations without any knowledge of a secret key, from a random permutation, which is a set of $2^n!$ n-bit permutations.

A secure block cipher should be resistant to both key recovery and distinguishing attacks within a practical time, assuming that the encryption/decryption algorithm is public. A block cipher is considered secure if there is no effective way with a fewer cost than an exhaustive search to recover the secret key or distinguish it from a random permutation. It should be mentioned that a

distinguishing attack does not directly lead to the recovery of the secret key. However, the success of a distinguishing attack implies that a block cipher has a non-randomness property (such a non-randomness property called *distinguisher*), and it may lead to the success of a key recovery attack in the end. In many cases, a key recovery attack is built on a certain distinguisher by extending the number of attacked rounds.

Since evaluating the longest round of distinguishers for any attack vector is a general approach to estimating the security of a new block cipher design, we focus on distinguishing attacks in this thesis.

2.3.2.2 For AEAD The primary purpose of attacks on AEADs is to violate data confidentiality and integrity. As mentioned in Section 2.2, block-cipher-based AEADs can claim their security followed by provable security, while dedicated AEADs, such as sponge-based AEADs, are hard to claim it on provable security. Therefore, these AEADs typically provide their security through a detailed analysis of existing attack vectors. In terms of data confidentiality, the security of sponge-based AEADs relies on the randomness of the keystream (output). To ensure data integrity, these AEADs must resist forgery attacks in which the attacker attempts to create a counterfeit authentication tag or a valid message with an original authentication tag. Before outlining the goal of attacks, we first provide the common requirement for the use of sponge-based AEADs.

Definition 7 **Nonce respecting:** *A nonce or IV is used only once to protect a message and associated data, meaning that the same pair of a nonce and secret key cannot be used to encrypt two different pairs of message and associated data.*

In the nonce-respecting setting, the attacker is only able to query a single ciphertext that corresponds to a chosen message with the same nonce. This means that the majority of queries in the attack procedure are for chosen ciphertexts.

Key recovery attack The attacker recovers a secret key of size k selected from any choice in the key space 2^k. Once a secret key is recovered, the attacker can do anything, such as recover plaintexts and internal states. This attack directly violates both data confidentiality and integrity.

State recovery attack The attacker recovers an internal state. In sponge-based AEADs, the permutation is often constructed with simple operations to maintain efficiency, and therefore an attacker may be able to recover the secret key by computing the inverse permutation once the internal state has been recovered. Hence, the success of the state recovery attack leads to violating both data confidentiality and integrity.

Distinguishing attack The attacker distinguishes the output from a random output in an encryption phase. It aims to violate a pseudo-randomness of an output, which leads to recover a plaintext.

Forgery attack The attacker produces a quartet of a ciphertext C, nonce N, associated data A, and authentication tag T that the decrytion oracle does not return $\perp$, with no encryption request of the same $\{N, A\}$ corresponding $\{C, T\}$. the success of the forgery attack leads to violating data integrity. Many AEADs assume that the attacker can produce the pair of a message and tag for any given message with a nonce-respecting setting. This type of forgery is called *universal forgery*.

A secure AEAD needs to resist these attacks with the assumption that an encryption/decryption algorithm is public as well as the case of a block cipher. An AEAD can be considered secure if there is no method for recovering the secret key or internal state except through an exhaustive search, and if a forgery attack requires a number of decryption queries greater than the security claim for such an attack.

2.3.3 Cryptanalysis Techniques

Currently, there are several techniques to analyze symmetric-key cryptographic algorithms. In this section, we introduce several techniques related to this thesis.

2.3.3.1 Differential Cryptanalysis The differential cryptanalysis is a chosen plaintext attack proposed by Biham and Shamir [43]. Nowadays, the differential cryptanalysis is the one of most popular and powerful cryptanalysis techniques available for symmetric-key cryptographic algorithms. The current American standard block cipher AES [5] was designed to have resistance against the differential cryptanalysis (and linear cryptanalysis), and all modern symmetric-key cryptographic algorithms not only block ciphers follow this principle. The idea of this technique is to exploit the difference between the pair of two plaintexts and the difference between the pair of the corresponding two ciphertexts. More specifically, in the differential cryptanalysis, the attacker attempts to find a pair of input and output differences with a high probability, i.e., $E_K(\Delta M) = \Delta C,\ (\Delta C = C \oplus C', \Delta M = M \oplus M')$ occurs with high probability on symmetric-key primitives E_k, where (M, M') and (C', C) denote a pair of plaintexts and ciphertexts, respectively. A pair of input and output differences $(\Delta M, \Delta C)$ is called a *differential* in the context of the differential cryptanalysis. The probability of a differential, which is called a *differential probability*, is calculated by investigating all pairs of plaintext following $\Delta M = M \oplus M'$ on E_K. We define a differential and its probability on a symmetric-key primitive E_K as follows.

Definition 8 **Differential** *A differential is a pair of input and output differences. The probability of a differential* $(\Delta P, \Delta C)$ *is calculated as follows:*

$$DP(\Delta M \xrightarrow{E_K} \Delta C) = \frac{\#\{x \in \mathbb{F}_2^n | E_K(M) \oplus E_K(M \oplus \Delta M) = \Delta C\}}{2^n},$$

where M *and* n *denote chosen from a uniformly distributed random variable and the size of block, respectively.*

In an ideal block cipher, the differential probability for any differential has to be 2^{-n}. Therefore, the attacker succeeds in the distinguishing attack if she obtains the differential with a probability more than 2^{-n}.

Generally, calculating the exact differential probability is computationally infeasible in most symmetric-key primitives. Therefore, a *differential characteristic* is usually employed to estimate a differential probability. Let E_K be a r-round iterated block cipher as $E_K(\cdot) = f_r(\cdot) \circ f_{r-1}(\cdot) \circ \cdots \circ f_1(\cdot)$. A differential characteristic can be defined as a sequence of differences over all rounds in E_K, and its probability can be estimated as a product of differential probabilities of each round under the well-known Markov cipher assumption.

Definition 9 **Markov cipher** [113] *An iterated cipher with round function* $x_o = F_R(x_i, rk)$*, where* x *and* k *denote the internal state and round key respectively, is a markov cipher if there is a group operation* $\oplus$ *for differences such that for all choices of* α *and* β $(\alpha \neq e, \beta \neq e)$*,* $DP(\Delta x_o = \beta | \Delta x_i = \alpha, x_i = \gamma)$ *is independent of* γ*, where a round key is uniformly random.*

We give the definition of a differential characteristic and its probability on a block cipher E_K as follows.

Definition 10 **Differential characteristic** *A differential characteristic is a sequence of differences over all rounds in a block cipher* E_K *as follows:*

$$\boldsymbol{C} = (\boldsymbol{c_0} \xrightarrow{f_1} \boldsymbol{c_1} \xrightarrow{f_2} \cdots \xrightarrow{f_r} \boldsymbol{c_r}) := (\boldsymbol{c_0}, \boldsymbol{c_1}, \cdots, \boldsymbol{c_r}),$$

where $(\boldsymbol{c_0}, \boldsymbol{c_1}, \cdots, \boldsymbol{c_r})$ *denotes differences of the output of each round, i.e.,* $\boldsymbol{c_0}$ *and* $\boldsymbol{c_r}$ *denote differences of plaintext and ciphertext, respectively. The probability of a differential characteristic* $\boldsymbol{C}$ *is estimated as follows:*

$$DP(\boldsymbol{C}) = \prod_{i=1}^{r} DP(\boldsymbol{c_{i-1}} \xrightarrow{f_i} \boldsymbol{c_i}).$$

From the perspective of the designer, it is sufficient to ensure that the upper bound of $DP(\boldsymbol{C})$ is achieved, rather than finding the optimal differential characteristic.

Since the probabilistic operations in symmetric-key primitives are generally an S-box and modular addition, and this thesis focuses on S-box-based primitives, we can estimate the maximum differential probability for a differential characteristic by counting the number of active S-boxes (AS), which is an S-box with a non-zero input difference. Let DP_s be the maximum differential probability of an S-box. We can estimate the upper bound of $DP(\boldsymbol{C})$ by the lower bound for the number of AS, i.e., $2^{-(DP_s \times \#AS)} \leq 2^{-n}$ is sufficient to resist against the distinguishing attack for any input (plaintext) difference.

Nowadays, it is common to use automatic search tools based on MILP, SAT/SMT, and CP to evaluate the optimal differential characteristic and the lower bound for the number of AS. In this thesis, we employ a MILP-based automatic search method described in Section 3.

2.3.3.2 Linear Cryptanalysis The linear cryptanalysis is a known plaintext attack proposed by Matsui [123]. With the linear cryptanalysis, Matsui broke the previous American standard block cipher DES, this cryptanalysis technique is considered to the two prime cryptanalysis techniques available for symmetric-key cryptographic algorithms as well as the differential cryptanalysis. The idea of this technique is to approximate the output Boolean function of symmetric-key primitives into one's input Boolean function as a probabilistic equation. More specifically, the attacker attempts to express the relation of the plaintext M, ciphertext C, and secret key K as a probabilistic equation, i.e., $\Gamma_\alpha(M) \oplus \Gamma_\beta(C) = \Gamma_\gamma(K)$ where Γ_α, Γ_β, and Γ_γ are called the liner masks. In many cases, we can see $\Gamma_\gamma(K)$ as a constant value $c \in \mathbb{F}_2$ since a secret is usually assumed as a fixed value. Hence, the main aim is to find Γ_α and Γ_β to satisfy $\Gamma_\alpha(M) \oplus \Gamma_\beta(C) = c$ with a high probability. This probability is called an *absolute linear bias* and defined as follows.

Definition 11 Absolute linear bias *An absolute linear bias with an input linear mask Γ_α and output linear mask Γ_β on an n-bit block symmetric-key primitive E_K is calculated as follows:*

$$LP(\Gamma_\alpha \xrightarrow{E_K} \Gamma_\beta) = \left(2\frac{\#\{M \in \mathbb{F}_2^n | \Gamma_\alpha(M) = \Gamma_\beta(C)\}}{2^n} - 1\right)^2$$

In an ideal n-bit block symmetric-key primitive, an absolute linear bias for any input and output mask has to be 2^{-n}. Therefore, the attacker succeeds in the distinguishing attack if she obtains the pair of input and output masks with an absolute linear bias more than 2^{-n}.

Generally calculating the exact absolute linear bias is computationally infeasible in most symmetric-key primitives. Therefore, a *linear characteristics* is usually employed to estimate an absolute linear bias, as well as the differential cryptanalysis. A linear characteristic can be defined as a sequence of the pair of input and output linear masks of each round in E_K, and its absolute linear bias can be estimated as a product of absolute linear biases of the pairs of each

round under Markov cipher assumption. The definition of a linear characteristic on a r-round iterated symmetric-key primitive $E_K(\cdot) = f_r(\cdot) \circ f_{r-1}(\cdot) \circ \cdots \circ f_1(\cdot)$ is given as follows.

Definition 12 **Linear characteristic** *A linear characteristic is a sequence of the pair of an input and output linear masks over all rounds in a block cipher E_K as follows:*

$$\boldsymbol{C_l} = (\boldsymbol{c_0^l} \xrightarrow{f_1} \boldsymbol{c_1^l} \xrightarrow{f_2} \cdots \xrightarrow{f_r} \boldsymbol{c_r^l}) := (\boldsymbol{c_0^l}, \boldsymbol{c_1^l}, \cdots, \boldsymbol{c_r^l}),$$

where $(\boldsymbol{c_0^l}, \boldsymbol{c_1^l}, \cdots, \boldsymbol{c_r^l})$ denotes linear masks of the output of each round, i.e., $\boldsymbol{c_0^l}$ and $\boldsymbol{c_r^l}$ denote linear masks of plaintext and ciphertext, respectively. The absolute linear bias of a linear characteristic $\boldsymbol{C_l}$ is estimated as follows:

$$LP(\boldsymbol{C_l}) = \prod_{i=1}^{r} LP(\boldsymbol{c_{i-1}^l} \xrightarrow{f_i} \boldsymbol{c_i^l}).$$

the perspective of the designer, it is also enough to guarantee the upper bound of $LP(\mathbf{C_l})$. Therefore, the active S-box-based evaluation is sufficient for the designer as well as in the case of the differential cryptanalysis. Let LP_s be the maximum absolute linear bias of an S-box. We can estimate the upper bound of $LP(\mathbf{C_l})$ by the lower bound for # AS, i.e., $2^{-(LP_s \times \#AS)} \leq 2^{-n}$ is sufficient to resist against the distinguishing attack for any input and output linear mask. This can be efficiently evaluated by automatic search tools as that of the differential cryptanalysis.

2.3.3.3 Impossible Differential Cryptanalysis

The impossible differential cryptanalysis is a chosen plaintext attack proposed by Biham et al. [41]. It is generally known that this cryptanalysis technique is one of the most powerful attacks to Feistel-based block ciphers. The main concept of the impossible differential cryptanalysis is to exploit a propagation property of the difference like the differential cryptanalysis. However, the differential cryptanalysis aims to find a pair $(\Delta M, \Delta C)$ which ΔM reaches ΔC with as high a probability as possible, while the impossible differential cryptanalysis aims to find a pair of input and output differences $(\Delta M, \Delta C)$ which ΔM never reaches ΔC after several rounds. Such differences are called the impossible differential distinguisher, and the attacker succeeds in the distinguishing attack if she obtain the impossible differential distinguishers.

Generally, the maximum number of rounds that the impossible differential distinguishers exist, the longest impossible differential distinguisher, can be roughly estimated by a *diffusion property*, which is one of the most important property when the designer design a new symmetric-key primitives.

Diffusion property A diffusion property is generally evaluated by the number of rounds to achieve *full diffusion*, which means that an input of any branch (bit) affects the outputs of

all branches (bits). To define the number of rounds to achieve the full diffusion, we define the number of rounds to achieve "diffusion" DR_n which means the number of rounds that the input of n-th branch affects the outputs of all branches. Fig 2.11 shows an example of the case of satisfying $DR_2 = 5$ on type-2 GFN. Let DR_{max} be the number of rounds to

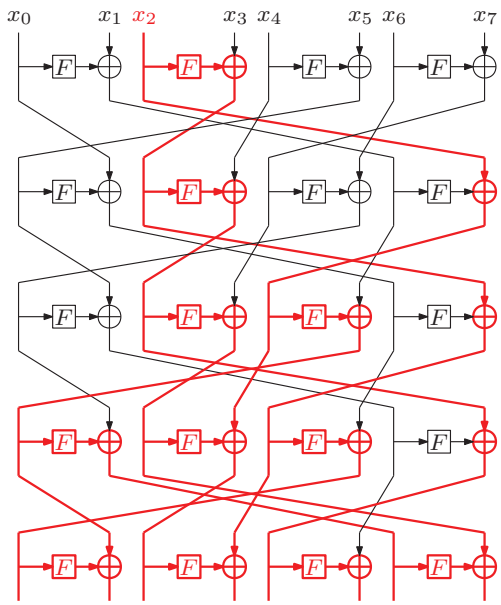


Figure 2.11: $DR_2 = 5$ on type-2 GFN. Red lines mean to be affected by the input difference of x_2.

achieve the full diffusion with the k-branch GFN. DR_{max} is defined as,

$$DR_{max} = \max_{x=0,\ldots,k-1} DR_x.$$

Generally, we evaluate DR_{max} from both encryption and decryption sides.

Roughly speaking, we can estimate the longest impossible differential distinguisher by a sum of DR_{max} of encryption and decryption directions. Therefore, the resistance against impossible differential attacks is getting stronger as the diffusion property is getting better.

2.3.3.4 Integral Cryptanalysis The integral cryptanalysis is a chosen plaintext attack proposed by Daemen et al. [63]. This cryptanalysis technique was first proposed as a dedicated attack to a block cipher Square, and then it was formalized to the *integral propert* by Knudsen and Wagner [110]. In the integral property, a set of plaintexts (also ciphertexts, and internal states) is divided into four property as follows:

ALL (A) The set contains all possible taken values the same number of times.

BALANCE (B) The XOR of all values in the set is zero.

CONSTANT (C) All values in the set are equal.

UNKNOWN (U) Each value in the set is random.

In an ideal symmetric-key primitive, any set of ciphertexts has to be **U**. Hence, the attacker succeeds the distinguishing attack if she find **B** in the set of certain ciphertexts.

At EUROCRYPT 2015, Todo further generalized the integral property into the *division property* [170] that can exploit the hidden feature between **A** and **B**. To give the definition of the division property, we first define the bit-product functions as follows.

Definition 13 Bit-product function *For any $u \in \mathbb{F}_2^n$, let $\pi_u(x)$ be a function from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. For any $x \in \mathbb{F}_2^n$, define $\pi_u(x)$ as follows:*

$$\pi_u(x) = \prod_{i=0}^{n-1} x[i]^{u[i]}.$$

Let $\boldsymbol{\pi_u}$ be a function from $(\mathbb{F}_2^{n_0} \times \mathbb{F}_2^{n_1} \times \ldots \times \mathbb{F}_2^{n_{m-1}})$ to $\mathbb{F}^2$ for all $\boldsymbol{u} \in \mathbb{F}_2^n$. For any $\boldsymbol{u} = (u_0, u_1, \ldots, u_{m-1}), \boldsymbol{x} = (x_0, x_1, \ldots, x_{m-1})$, define $\boldsymbol{\pi_u}(\boldsymbol{x})$ as follows.

$$\boldsymbol{\pi_u}(\boldsymbol{x}) = \prod_{i=0}^{m-1} \pi_{u_i}(x_i).$$

The division property is defined as follows with the bit-product function.

Definition 14 Division property *Let $\mathbb{X}$ be a multiset whose elements take a value of $(\mathbb{F}_2^{n_1} \times \mathbb{F}_2^{n_2} \times \ldots \times \mathbb{F}_2^{n_m})$. When the multiset $\mathbb{X}$ has the division property $\mathcal{D}_{\mathbb{K}}^{n_1,\ldots,n_m}$, where $\mathbb{K}$ denotes a set of m-dimensional vectors whose i-th element takes 0 and n_i, it fulfills the following conditions.*

$$\bigoplus_{\boldsymbol{x} \in \mathbb{X}} \pi_{\boldsymbol{u}}(\boldsymbol{x}) = \begin{cases} unknown & \text{if there exist } \boldsymbol{k} \in \mathbb{K}\ s.t\ wt(\boldsymbol{u}) \succeq \boldsymbol{k}, \\ 0 & otherwise. \end{cases}$$

$wt(\boldsymbol{u})$ is the Hamming weight of $\boldsymbol{u}$. If there exist $\boldsymbol{k} \in \mathbb{K}$ and $\boldsymbol{k}' \in \mathbb{K}$ satisfying $\boldsymbol{k} \succeq \boldsymbol{k}'$ in the division property $\mathcal{D}_{\mathbb{K}}^{n_1,\ldots,n_m}$, $\boldsymbol{k}$ can be removed from $\mathbb{K}$ because it is redundant.

For a better understanding, we give several propagation rules of the division property over basic operations related to this thesis.

Lemma 1 XOR *Let $(\boldsymbol{x_0}, \boldsymbol{x_1}) \in (\mathbb{F}_2^n \times \mathbb{F}_2^n)$ and $\boldsymbol{y}$ be the input and output of an XOR operation, respectively, i.e., $\boldsymbol{y} = \boldsymbol{x_0} \oplus \boldsymbol{x_1}$. When the division property of the set of the input $\mathbb{X}$ is $\mathcal{D}_{\mathbb{K}}^{n,n}$,*

the division property $\mathcal{D}^n_{k'}$,which the division property of the set of the output $\mathbb{Y}$, is calculated as follows:

$$k' = \min_{(k_1,k_2)\in\mathbb{K}} \{k_1 + k_2\},$$

where $k' \leq n$.

Lemma 2 COPY (forked branch) *Let $\boldsymbol{x} \in \mathbb{F}_2^n$ and $(\boldsymbol{y_0}, \boldsymbol{y_1}) \in (\mathbb{F}_2^n \times \mathbb{F}_2^n)$ be the input and output of a copy operation, respectively, i.e., $(\boldsymbol{y_0}, \boldsymbol{y_1}) = (\boldsymbol{x}, \boldsymbol{x})$. When the division property of the set of the input $\mathbb{X}$ is $\mathcal{D}^n_{\mathbb{K}}$, the division property $\mathcal{D}^{n,n}_{\mathbb{K}'}$,which the division property of the set of the output $\mathbb{Y}$, is calculated as follows:*

$$\mathbb{K}' = \mathbb{K}' \cup (k - i, i),\ \forall\, i\ (0 \leq i \leq k).$$

Lemma 3 Parallel application of S-boxes *Suppose the size and the algebraic degree of i-th S-box are n_i and d_i, respectively. When the division property of the set of the input $\mathbb{X} \in (\mathbb{F}_2^{n_1} \times \mathbb{F}_2^{n_2} \times \cdots \times \mathbb{F}_2^{n_m})$ is $\mathcal{D}^{n_1,n_2,\ldots,n_m}_{\mathbb{K}}$, the division property of the set of the output $\mathcal{D}^{n_1,n_2,\ldots,n_m}_{\mathbb{K}'}$ is calculated as follows:*

$$\mathbb{K}' = \mathbb{K}' \cup (\lceil \frac{k_1}{d_1} \rceil, \lceil \frac{k_2}{d_2} \rceil, \ldots, \lceil \frac{k_m}{d_m} \rceil),$$

where $(k_1, k_2, \ldots, k_m) \in \mathbb{K}$. Note that the output division property of the i-th S-box is equal to the input division property if and only if the i-th S-box is bijective and the input division property is $\mathcal{D}^{n_i}_{n_i}$.

With these propagation rules, we can view the division property in the output (ciphertext) that comes from the division property in the input (plaintext). When there is no division property $\mathcal{D}^{n_1,n_2,\ldots,n_m}_{\mathbb{K}'}$ of the set of output $\mathbb{Y}$, which satisfies $k_i = 1, k_j = 0 (\forall j, j \neq i)$, it means that i-th part has balance property. In general, the resistance against integral attacks is getting stronger as the diffusion property is getting better as well as in the case of the impossible differential cryptanalysis.

2.3.3.5 Internal Collision Based Forgery For sponge-based AEADs, the main threat to violate data integrity is the internal collision based forgery. The designer of a sponge-based AEAD has to design its round function with the resistance against this attack.

Let the encryption phase of a sponge-based AEAD be $F_k = f_r \circ f_{r-1} \circ \cdots \circ f_1$ where f_i denotes the i-th round round function. The internal collision happens in the output of the t-th round if $f_t(m_{0,t}) \circ f_{t-1}(m_{0,t-1}) \circ \cdots \circ f_1(m_{0,1}) = f_t(m_{1,t}) \circ f_{t-1}(m_{1,t-1}) \circ \cdots \circ f_1(m_{1,1})$

where $M_i = (m_{i,1}, m_{i,2}, \ldots m_{i,t})$ denotes a message. In other words, it can be viewed as $f_t(\Delta m_t) \circ f_{t-1}(\Delta m_{t-1}) \circ \cdots \circ f_1(\Delta m_1) = 0$ where $M_0 \oplus M_1 = \Delta M = (\Delta m_1, \Delta m_2, \ldots \Delta m_t)$. Since the finalization phase does not accept any input, the attacker can produce the valid pair of the ciphertext and authentication tag without knowledge of a secret key if she finds the internal collision.

When the tag length is l_t bits, the probability that the attacker coincidentally produces a valid pair is 2^{-l_t}. Therefore, the attacker succeeds in a forgery attack if they are able to produce such a valid pair with a probability greater than 2^{-l_t}.

In an internal collision-based forgery, the success probability is equal to the probability of an internal collision occurring. This probability can be expressed as the differential probability of the differential propagation that non-active first round internal state differences reach non-active last round internal state differences in the encryption phase.

More specifically, the attacker attempts to find a differential propagation with a high probability in the encryption phase, given that all first round state differences and the output of the last round state differences are set to zero. The security evaluation of this attack can be conducted using the same method as differential cryptanalysis, with the aid of automatic search tools.

3 MILP-Aided Security Evaluation

Mixed Integer Linear Programming (MILP) is one of the algorithms for efficiently solving an optimization problem. An MILP model $\mathcal{M}$ is constructed by variables $\mathcal{M}_{var}$, constraints $\mathcal{M}_{con}$, and the objective function $\mathcal{M}_{obj}$. We can obtain the optimized solution by giving an MILP model to an MILP solver. We here give a brief example of an MILP as follows:

$$\begin{cases} \mathcal{M}var & \leftarrow x, y, z, \\ \mathcal{M}con & \leftarrow x + 2y + 3z \leq 4, \quad x + y \leq 1, \\ \mathcal{M}_{obj} & \leftarrow Maximize(x + y + 2z). \end{cases}$$

This example shows an optimization problem while we can also solve a feasibility problem, the problem to judge if there is a solution satisfying all constraints, by removing the objective function.[3] If a given MILP model has at least one solution, a solver returns one solution among all of them, and if there is no solution, a solver returns ”infeasible”.

Nowadays, many automatic security evaluation tools with an MILP have been proposed for the latest decade, since then Mouha et al.proposed the first automatic security evaluation method of differential/linear cryptanalysis by an MILP. The advancement of automatic tools has enabled us to evaluate security against a wide range of attack vectors, including differential, linear, integral, and impossible differential cryptanalysis. These tools offer several benefits compared to traditional heuristic evaluation methods, most notably a reduction in evaluation runtime. Through the use of MILP methods, we are able to perform complex security evaluations in a practical time, something that is not possible with conventional methods. The development of automatic evaluation methods utilizing mathematical solvers has greatly impacted the field of symmetric-key cryptography.

To utilize an MILP in security evaluation, we need to translate the propagation of a difference, linear mask, and division property over all operations in algorithms into linear inequalities and incorporate them into the MILP model as constraints. The objective function is then set and the MILP solver is ready to perform the evaluation. In the remainder of this section, we elaborate on how to construct MILP models for evaluating security against specific cryptanalysis techniques. In this thesis, we employ Gurobi optimization solver as our MILP solver for all security evaluations.

[3] As one of the method to efficiently solve a kind of a feasibility problem, *Boolean satisfiability problem*(SAT) is often mentioned, which is another popular automatic evaluation method in the field of symmetric-key cryptography.

3.1 Security Evaluation of Differential/Linear Cryptanalysis

It is known that the differential and linear cryptanalysis have duality results [56], meaning that we can use a nearly identical approach to construct an MILP model for evaluating security against both types of attacks. Therefore, we present both MILP models in comparison.

Before delving into the specific models, we first introduce the concept of a *truncated difference/linear mask*. To evaluate security against differential/linear cryptanalysis, it is necessary to ensure that there are no differential/linear characteristics with a probability greater than 2^{-n}, where n is the block size or claimed security level. In a known/chosen plaintext scenario, an attacker can construct an arbitrary difference/linear mask in the plaintext. As a result, the designer must find the differential/linear characteristic with the highest probability under any possible plaintext difference/linear mask. This can be computationally infeasible, particularly for block sizes larger than 32 bits, as the plaintext space exceeds $2^{32} - 1$. To address this issue, designers often utilize a truncated difference/linear mask, in which a difference/linear mask is treated in units such as nibbles or bytes rather than bits. An example of a 4-bit (nibble-wise) truncated difference/linear mask is as follows:

$$\Delta x = \begin{cases} 0 & \text{if } (\Delta x_0, \Delta x_1, \Delta x_2, \Delta x_3) = (0,0,0,0), \\ 1 & \text{otherwise.} \end{cases}$$

$$\Gamma x = \begin{cases} 0 & \text{if } (\Gamma x_0, \Gamma x_1, \Gamma x_2, \Gamma x_3) = (0,0,0,0), \\ 1 & \text{otherwise.} \end{cases}$$

An 8-bit (byte-wise) truncated version can be represented by extending the above equations. Many designs are implemented as word-oriented designs, meaning that all operations in a design are performed in a word-wise manner, to facilitate security evaluation with a truncated difference/linear mask.

In the following sections, we describe the method for modeling the security evaluation of differential/linear cryptanalysis as proposed by Mouha et al. [133] and Sun et al. [165].

3.1.1 Constraints

As constraints in an MILP model, we must generate linear inequalities that describe the transitions of differences/linear masks over an algorithm. We can decompose the entire transition into operation-specific propagation rules. We can then express an entire transition over an algorithm as a set of linear inequalities of these operations. In this section, we introduce the propagation rules for an XOR, COPY, and S-box operations.

Lemma 4 XOR (differential) *Let (x_0, x_1) and y be inputs and output differences of an XOR operation, i.e., $x_0 \oplus x_1 = y$. Linear inequalities of a bit-wise differential propagation and its variables can be expressed as follows:*

$$\begin{cases} \mathcal{M}_{var} & \leftarrow x_0.x_1, \text{ as binary}, \\ \mathcal{M}_{con} & \leftarrow \begin{cases} x_0 + x_1 + y = 2d, \\ d \geq x_0, \\ d \geq x_1, \\ d \geq y, \end{cases} \end{cases}$$

where $d \in \mathbb{F}_2$ is a dummy variable. Besides, linear inequalities of a word-wise differential propagation and its variables can be expressed as follows:

$$\begin{cases} \mathcal{M}_{var} & \leftarrow x_0, x_1, \text{ as binary}, \\ \mathcal{M}_{con} & \leftarrow \begin{cases} -x_0 + x_1 + y & \geq 0, \\ x_0 - x_1 + y & \geq 0, \\ x_0 + x_1 - y & \geq 0. \end{cases} \end{cases}$$

Lemma 5 XOR (linear mask) *Let (x_0, x_1) and y be inputs and output linear masks of an XOR operation, i.e., $x_0 \oplus x_1 = y$. Linear inequalities of both bit and word-wise linear mask propagation and their variables can be expressed as follows:*

$$\begin{cases} \mathcal{M}_{var} & \leftarrow x_0, x_1, \text{ as binary}, \\ \mathcal{M}_{con} & \leftarrow x_0 = x_1 = y. \end{cases}$$

Lemma 6 COPY (differential) *Let x and (y_0, y_1) be the input and outputs of a COPY operation, i.e., $(y_0, y_1) = (x, x)$. Linear inequalities of both bit and word-wise differential propagation and their variables can be expressed as follows:*

$$\begin{cases} \mathcal{M}_{var} & \leftarrow x_0, x_1, \text{ as binary}, \\ \mathcal{M}_{con} & \leftarrow x_0 = x_1 = y. \end{cases}$$

Lemma 7 COPY (linear mask) *Let x and (y_0, y_1) be the input and outputs of a COPY operation, i.e., $(y_0, y_1) = (x, x)$. Linear inequalities of a bit-wise linear mask propagation and its*

variables can be expressed as follows:

$$\begin{cases} \mathcal{M}_{var} & \leftarrow x_0.x_1, \text{ as binary}, \\ \mathcal{M}_{con} & \leftarrow \begin{cases} x_0 + x_1 + y = 2d, \\ d \geq x_0, \\ d \geq x_1, \\ d \geq y, \end{cases} \end{cases}$$

where $d \in \mathbb{F}_2$ is a dummy variable. Besides, linear inequalities of a word-wise linear mask propagation and its variables can be expressed as follows:

$$\begin{cases} \mathcal{M}_{var} & \leftarrow x_0, x_1, \text{ as binary}, \\ \mathcal{M}_{con} & \leftarrow \begin{cases} -x_0 + x_1 + y & \geq 0, \\ x_0 - x_1 + y & \geq 0, \\ x_0 + x_1 - y & \geq 0. \end{cases} \end{cases}$$

For an S-box operation, *H-representation of convex hull* [165] and *logical condition model* [4] are known as the method to model the propagation. In this thesis, we employ a logical condition model. Since a bit-wise model for an S-box operation is very complex compared to the other models, we describe a bit and word-wise model individually.

Lemma 8 S-box (bit-wise differential) *Let $\boldsymbol{a} = (a_0, a_1, \ldots, a_{i-1})$ and $\boldsymbol{b} = (b_0, b_1, \ldots, b_{i-1})$ be the input and output differences of an i-bit S-box, respectively. Additionally, we must introduce additional Boolean variables s to judge whether an S-box corresponding to a differential propagation is active or not. With these Boolean variables, we construct the following Boolean formula:*

$$f(\boldsymbol{a}, \boldsymbol{b}, s) = \begin{cases} 1 \ \text{ if } \boldsymbol{a}(\neq \boldsymbol{0}) \overset{S-box}{\rightarrow} \boldsymbol{b} \text{ is valid with } s = 1 \text{ or } \boldsymbol{a}(= \boldsymbol{0}) \overset{S-box}{\rightarrow} \boldsymbol{b} \text{ is valid with } s = 0, \\ 0 \ \text{ otherwise}. \end{cases}$$

Then, we extract a set A, which contains all vectors satisfying $f(\boldsymbol{x}, \boldsymbol{y}, z) = 0$ as follows:

$$A = \{(\boldsymbol{x}, \boldsymbol{y}, z) \in \mathbb{F}_2^{2i+1} \mid f(\boldsymbol{x}, \boldsymbol{y}, z) = 0\},$$

where $\boldsymbol{x} = (x_0, x_1, \ldots, x_{i-1})$ and $\boldsymbol{y} = (y_0, y_1, \ldots, y_{i-1})$. Because A is a set of invalid patterns in a model of an S-box, we ban these patterns by the following clauses:

$$\bigvee_{c=0}^{i-1} (a_c \oplus x_c) \vee \bigvee_{d=0}^{i-1} (b_d \oplus y_d) \vee (s \oplus z) = 1, \ (\boldsymbol{x}, \boldsymbol{y}, z) \in A.$$

The remaining vectors that are identical to $\overline{A}$ are a set of valid patterns. Therefore, these clauses extract the differential propagation with the status of an i-bit S-box. Note that the solution space of $|A|$ clauses about $(\boldsymbol{a}, \boldsymbol{b}, s)$ in Eq. (3.2) is identical to that of the following Boolean function:

$$g(\boldsymbol{a}, \boldsymbol{b}, s) = \bigwedge_{\eta=0}^{|A|-1} \left(\bigvee_{c=0}^{i-1} (a_c \oplus x_c^\eta) \vee \bigvee_{d=0}^{i-1} (b_d \oplus y_d^\eta) \vee (s \oplus z^\eta) \right) = 1.$$

This Boolean function is equivalent to

$$g(\boldsymbol{a}, \boldsymbol{b}, s) = \bigwedge_{(\boldsymbol{x},\boldsymbol{y},z) \in \mathbb{F}_2^{2i+1}} \left(g(\boldsymbol{x}, \boldsymbol{y}, z) \vee \bigvee_{c=0}^{i-1} (a_c \oplus x_c^\eta) \vee \bigvee_{d=0}^{i-1} (b_d \oplus y_d^\eta) \vee (s \oplus z^\eta) \right).$$

This equation is called the $product\text{-}of\text{-}sum$ of g. The issue of reducing the number of clauses in g is turned into the issue of simplifying the product-of-sum representation of the Boolean function. Owing to previous works [120, 163, 164], we know that this can be solved by the Quine-McCluskey algorithm [125, 145, 146] with the minimum expression. When this problem is relatively small, we can solve it by software, such as $Logic\ Friday$[4], although it is NP-complex. After applying Quine-McCluskey algorithm, we can obtain some Boolean clauses to express the differential propagation over an i-bit S-box. These clauses can be easily converted into the same number of linear inequalities, namely, every clause can be converted into one linear inequality. For example, when one of clauses is $(x_0 \vee x_2 \vee \overline{x_3} \vee \overline{y_1} \vee y_2)$, it can be converted into $x_0 + x_2 + (1 - x_3) + (1 - y_1) + y_2 \geq 1$. Therefore, linear inequalities of a bit-wise differential propagation and its variables can be expressed as follows:

$$\mathcal{M}_{var} \leftarrow (a_0, a_1, \ldots, a_{i-1}, b_0, b_1, \ldots, b_{i-1}, s),$$
$$\mathcal{M}_{con} \leftarrow min\left(g(\boldsymbol{a}, \boldsymbol{b}, s)\right).$$

Lemma 9 **S-box (word-wise differential)** *Let a and b be the input and output of an S-box, i.e., $x \xrightarrow{S-box} y$. Liner inequalities of a word-wise differential propagation and its variables can be expressed as follows:*

$$\begin{cases} \mathcal{M}_{var} & \leftarrow x, y, \text{ as binary}, \\ \mathcal{M}_{con} & \leftarrow x = y. \end{cases}$$

Since the propagation of a linear mask through an S-box can be represented by linear inequalities in the same manner as differential propagation, we will not further discuss it.

[4] `http://www.sontrak.com/`

3.1.2 Objective Function

As described in Sect. 2.3.3.1 and 2.3.3.2, the security evaluation against the differential/linear cryptanalysis is often assessed by the number of active S-boxes (# AS). In the AS-based evaluation, the designer need to guarantee that the lower bound for the number of active S-boxes outnumbers a certain bound to resist these attacks. Hence, we can directly obtain the lower bound for the number of active S-boxes by minimizing the number of active S-boxes as the objective function. In a bit-wise model, we give an MILP model the following objective function:

$$\sum_{i=1}^{r}\sum_{j=0}^{n-1} s_{i,j},$$

where r, n, and $s_{i,j}$ denote the number of rounds, the number of S-boxes in a round, and Boolean variables to judge whether an S-box corresponding to a differential propagation is active or not in Lemma 8, respectively.

In a word-wise model, we can tell whether an S-box is active or not by seeing the status of the input variable of an S-box. If it is active (=1), an S-box is active. Otherwise, an S-box is inactive. Therefore, we give an MILP model the following objective function.

$$\sum_{i=1}^{r}\sum_{j=0}^{n-1} x_{i,j},$$

where $x_{i,j}$ denotes the input variable of each S-box.

Now, we are ready to conduct the AS-based security evaluation. By giving the constructed MILP model $(\mathcal{M}_{var}, \mathcal{M}_{con}, \mathcal{M}_{obj})$ to an MILP solver and minimizing the objective function, we can obtain the lower bound for the number of active S-boxes and evaluate a resistance against the differential/linear cryptanalysis.

Note that we can evaluate the security against the internal collision based forgery by the same method as that of the differential cryptanalysis because it is also evaluated by active S-box based evaluation.

3.2 Security Evaluation of Impossible differential Cryptanalysis

To assess resistance to impossible differential cryptanalysis, it is necessary to evaluate the longest-round impossible differential distinguisher. When constructing the MILP model to evaluate security against this type of attack, we can reuse the same variables and constraints as in the MILP model to evaluate security against the differential cryptanalysis, as this attack involves tracing differential propagation over the entire algorithm. However, the MILP model for impossible differential cryptanalysis differs in that it is a feasibility problem rather than an optimization problem, meaning that it does not have an objective function.

Instead of giving the objective function, we fix the input (plaintext) and output (ciphertext) differences and check whether an MILP model has a solution or not. If an MILP model has a solution (feasible), it indicates that there exists a transition from the fixed plaintext differences to the fixed ciphertext differences, and therefore there are no impossible differential distinguishers for the given input and output differences. The longest impossible differential distinguisher can be evaluated by conducting this process for all combinations of input and output differences.

3.3 Security Evaluation of Integral Cryptanalysis

It is know that the most efficient approach to evaluate the longest integral distinguisher is to investigate it by the division property. As a method to efficiently evaluate the division property, Xiang et al. proposed how to evaluate the division property with the *division trail* by a MILP, which allows us to illustrate the propagation of the division property and makes the evaluation much easier [177]. The division trail is defined as follows.

Definition 15 **Division trail** *Let f_r denote the round function of an iterated block cipher. Assume the input multiset to the block cipher has initial division property $\mathcal{D}_{\boldsymbol{k}}^{n,m}$, and denote the division property after i-round propagation through f_r by $\mathcal{D}_{\mathbb{K}_i}^{n,m}$. Thus, we have the following chain of division property propagations:*

$$\{\boldsymbol{k}\} \stackrel{def}{=} \mathbb{K}_0 \xrightarrow{f_r} \mathbb{K}_1 \xrightarrow{f_r} \mathbb{K}_2 \xrightarrow{f_r} \cdots .$$

Moreover, for any vector $\boldsymbol{k}_i^*$ in $\mathbb{K}_i (i \geq 1)$, there must exist an vector $\boldsymbol{k}_{i-1}^*$ in $\mathbb{K}_{i-1}$ such that $\boldsymbol{k}_{i-1}^*$ can propagate to $\boldsymbol{k}_i^*$ by division property propagation rules. Furthermore, for $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r) \in \mathbb{K}_0 \times \mathbb{K}_1 \times \ldots \times \mathbb{K}_r$, if $\boldsymbol{k}_{i-1}$ can propagate to $\boldsymbol{k}_i$ for all $i \in \{1, 2, \ldots, r\}$, we call $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r)$ an r-round division trail.

Proposition 1 *Denote the division property of input multiset to an iterated block cipher by $\mathcal{D}_{\boldsymbol{k}}^{n,m}$, let f_r be the round function. Denote*

$$\{\boldsymbol{k}\} \stackrel{def}{=} \mathbb{K}_0 \xrightarrow{f_r} \mathbb{K}_1 \xrightarrow{f_r} \mathbb{K}_2 \xrightarrow{f_r} \cdots \xrightarrow{f_r} \mathbb{K}_r.$$

the r-round division property propagation. Thus, the set of the last vectors of all r-round division trails which start with $\boldsymbol{k}$ is equal to $\mathbb{K}_r$.

If any vector of $\mathbb{K}_r$ that is derived from the division property $\mathcal{D}_{\mathbb{K}_0}$ of input multiset is always less than or equal to "1", it means that there is no integral distinguisher at r rounds.

3.3.1 Constraints

To catch the division trail by an MILP, we need to convert the propagation of the division property into linear inequalities and give them to an MILP model as constraints. It can be decomposed operation by operation as well as that of the differential/linear cryptanalysis. Hence, we describe how to convert the propagation of the division property over an XOR, COPY, S-box operations.

Lemma 10 **XOR** *Let $(\boldsymbol{x_0}, \boldsymbol{x_1}) \in (\mathbb{F}_2^n \times \mathbb{F}_2^n)$ and $\boldsymbol{y}$ be the input and output of an XOR operation, respectively, i.e., $\boldsymbol{y} = \boldsymbol{x_0} \oplus \boldsymbol{x_1}$. When the division property of the set of the input $\mathbb{X}$ is $\mathcal{D}_{\mathbb{K}}^{n,n}$, by Lemma 1, we kwon that the division property $\mathcal{D}_{k'}^{n}$, which the division property of the set of the output $\mathbb{Y}$, is calculated by $k' = \min_{(k_1,k_2)\in\mathbb{K}} \{k_1 + k_2\}$ where $k' \leq n$. Therefore, linear inequalities of the propagation of the division property and its variables can be expressed as follows:*

$$\begin{cases} \mathcal{M}_{var} & \leftarrow k_1, k_2, k' \; as \; integer \; (0 \leq k_1, k_2, k' \leq n), \\ \mathcal{M}_{con} & \leftarrow k_1 + k_2 = k'. \end{cases}$$

Lemma 11 **COPY (forked branch)** *Let $\boldsymbol{x} \in \mathbb{F}_2^n$ and $(\boldsymbol{y_0}, \boldsymbol{y_1}) \in (\mathbb{F}_2^n \times \mathbb{F}_2^n)$ be the input and output of a copy operation, respectively, i.e., $(\boldsymbol{y_0}, \boldsymbol{y_1}) = (\boldsymbol{x}, \boldsymbol{x})$. When the division property of the set of the input $\mathbb{X}$ is $\mathcal{D}_{\mathbb{K}}^{n}$, by Lemma 2, we know that the division property $\mathcal{D}_{\mathbb{K}'}^{n,n}$, which the division property of the set of the output $\mathbb{Y}$, is calculated by $\mathbb{K}' = \mathbb{K}' \cup (k-i, i), \; \forall \, i \; (0 \leq i \leq k)$. Therefore, linear inequalities of the propagation of the division property and its variables can be expressed as follows:*

$$\begin{cases} \mathcal{M}_{var} & \leftarrow k, k_1', k_2' \; as \; integer \; (0 \leq k, k_1', k_2' \leq n), \; (k_1', k_2') \in \mathbb{K}', \\ \mathcal{M}_{con} & \leftarrow k_1' + k_2' = k. \end{cases}$$

Lemma 12 **S-box** *Suppose the size and the algebraic degree of i-th S-box are n_i and d_i, respectively. When the division property of the set of the input $\mathbb{X} \in (\mathbb{F}_2^{n_1} \times \mathbb{F}_2^{n_2} \times \cdots \times \mathbb{F}_2^{n_m})$ is $\mathcal{D}_{\mathbb{K}}^{n_1,n_2,\ldots,n_m}$, by Lemma 3, we know that the division property of the set of the output $\mathcal{D}_{\mathbb{K}'}^{n_1,n_2,\ldots,n_m}$ is calculated by $\mathbb{K}' = \mathbb{K}' \cup (\lceil \frac{k_1}{d_1} \rceil, \lceil \frac{k_2}{d_2} \rceil, \ldots, \lceil \frac{k_m}{d_m} \rceil)$, where $(k_1, k_2, \ldots, k_m) \in \mathbb{K}$, and the output division property of the i-th S-box is equal to the input division property if and only if the i-th S-box is bijective and the input division property is $\mathcal{D}_{n_i}^{n_i}$. Therefore, linear inequalities of the propagation of the division property and its variables over single n_i-bit S-box can be expressed as follows:*

$$\begin{cases} \mathcal{M}_{var} & \leftarrow k_i, k_i' \; as \; integer \; (0 \leq k_i, k_i' \leq n_i), \; k_i' \in \mathbb{K}', \\ \mathcal{M}_{con} & \leftarrow \begin{cases} k_i' = \frac{k_i}{d_i}, \\ k_i' \geq n_i k_i - n_i(n_i - 1). \end{cases} \end{cases}$$

3.3.2 Objective Function

Evaluating security against integral cryptanalysis is similar to evaluating security against impossible differential cryptanalysis in that it involves a feasibility problem rather than an optimization problem. Therefore, no objective function is given to an MILP model. In this context, the focus is on determining whether there exists a division trail from a given input division property to the output divsion property $\mathcal{D}_1^n$, where n is the block size. If such a trail exists, it means that there are no integral distinguishers. Generally, the designer need to evaluate its securit under the assumption of the attacker's potential ability to make full use of their resources. Therefore, the designer needs to evaluate an MILP model whose the input (plaintext) and output (ciphertext) division property are $\mathcal{D}_{n-1}^n$ and $\mathcal{D}_1^n$, respectively.

4 Tweakable TWINE: Building a Tweakable Block Cipher on Generalized Feistel Structure

Tweakable block cipher (TBC) is an extension of conventional block cipher. In this section, we study how to build a TBC based on generalized Feistel structure (GFS), a classical block cipher construction. While known dedicated TBC proposals are based on substitution-permutation network (SPN), GFS has not been used for building TBC. In particular, we take 64-bit GFS block cipher TWINE and try to make it tweakable with a minimum change. To find a best one from a large number of candidates, we performed a comprehensive search with a help of mixed integer linear programming (MILP) solver. As a result, our proposal Tweakable TWINE is quite efficient, has the same number of rounds as TWINE with extremely simple tweak schedule.

4.1 Introduction

4.1.0.1 Tweakable Block Cipher. Tweakable block cipher (TBC) is an extension of a conventional block cipher. An encryption of TBC is a function takes a public input called tweak T in addition to key K and plaintext M, and the pair (K, T) specifies the permutation over the message space. Since its inception by Liskov et al. [118] TBC has been extensively studied, and now it is widely acknowledged as a powerful primitive to build efficient and highly-secure symmetric-key modes of operations. For example, the seminal OCB authenticated encryption [112, 149, 150] scheme can be seen as a mode of TBC with TBC instantiated as a block cipher mode called XEX [149].

As proposed by Liskov et al., TBCs can be built on block ciphers. Typical examples are LRW and XEX modes of operations. They are efficient, as it requires few block cipher calls for one TBC encryption/decryption. For security, they have provable security guarantee up to around $2^{n/2}$ queries for TBC of n-bit block. Unfortunately, this level of security is not always enough, in particular when a TBC is used by modes of operation achieving "beyond-the-birthday-bound" (BBB) security. Some modes of operations to achieve BBB security have been proposed [114, 116, 173], however, they are usually much more costlier than the simple LRW/XEX.

Another approach, which is our focus, is a dedicated construction. To our knowledge the earliest proposal is HPC (where tweak is called "spice") proposed for AES competition [147]. In recent years, dedicated TBCs are becoming popular, such as Threefish [136], Deoxys-BC [106], SKINNY [27] and QARMA [9]. One strong advantage of dedicated construction is one can expect the full security when properly designed. While dedicated TBCs are possible in principle, by somehow absorbing tweak in the ordinary block cipher structures such as SPN or Feistel, but

the challenge is how to make it efficient, in a small number of rounds, while keeping the sufficient security. Tweakey framework [105] is one prominent methodology to SPN-based dedicated TBCs, and elastic-tweak framework [57] is proposed as a methodology to build TBCs having short-tweak based on SPN-based block cipher. However, in general, the constructions of TBCs are far less studied than those for block ciphers or fixed-key permutations.

4.1.0.2 Building TBC on GFS. In this section, we study how to build a TBC on generalized Feistel structure (GFS) [140, 179], one of the classical structures for block ciphers. GFS has been adopted by a number of block cipher proposals [94, 168]. However, it has never been used to build a TBC, to the best of our knowledge. Goldenberg et al. [84] and Mitsuda and Iwata [131] studied GFS-based TBC constructions from the viewpoint of provable security, where the round functions are instantiated by a pseudo-random functions (PRFs) and the focus is to evaluate a pseudo-random permutation (PRP) security of GFS. While these results give us a baseline, they tell little about the design and security of concrete TBCs. GFS has a large freedom regarding the choice of sub-block permutation. Suzaki and Minematsu [167] showed a comprehensive study on the effect of permutation, including the diffusion, the number of differential/linear active S-boxes, etc. Subsequently Suzaki et al. proposed 64-bit block cipher TWINE [168]. It uses a permutation over 16 nibbles selected from the result of [167] for achieving the best characteristics.

To build a TBC on GFS, we set our primary goal to reduce the cost of design, security evaluation, and implementation. Consequently, we choose to reuse TWINE as much as possible. We design an extremely simple tweak scheduling based on SKINNY's tweakey schedule [27] and attach it to TWINE. This reusing approach to dedicated TBC is useful for both designers and users, and we think our approach itself has some novelties (of course, some of the aforementioned work convert a block cipher into a TBC, but they are provably secure constructions). We evaluated linear/differential/impossible/integral characteristics for single-key, related-key and chosen-tweak setting. For finding the best parameter of tweak schedule against these attacks, we extensively used Mixed Integer Linear Programming (MILP) solver, which is now quite common for designing ciphers. With these efforts, our proposal, called Tweakable TWINE (or T-TWINE), becomes quite efficient : it has 64-bit tweak (the same as the block size) and has *the same number of 36 rounds* as the original. Key schedule and tweak schedule are independent, which will be useful for some use cases, e.g, when a key is hardwired. Tweakable TWINE is obtained by adding few nibble XORs to TWINE, therefore the hardware cost is essentially the same as TWINE except the registers for tweak. We also show basic hardware implementation results to verify this claim.

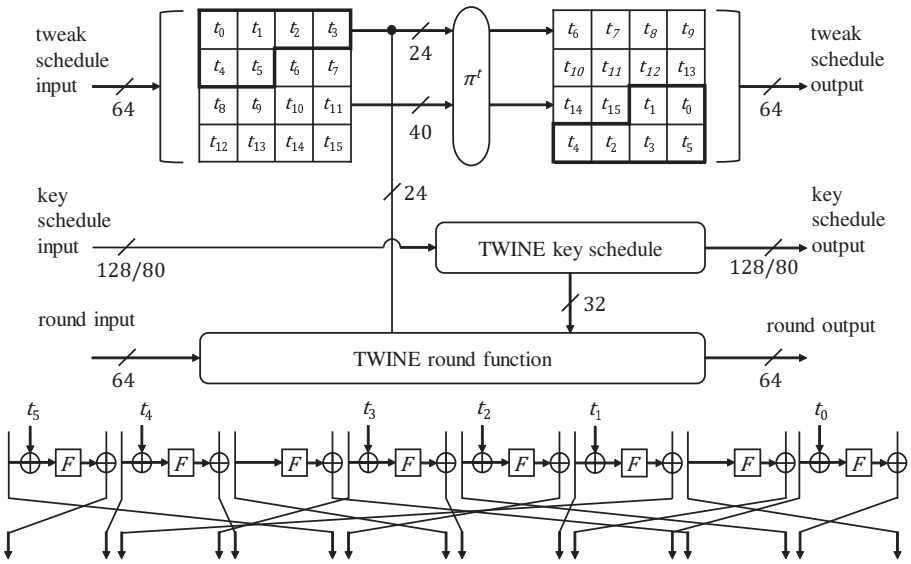


Figure 4.1: Overview of Tweakable TWINE

4.1.1 Organization of this section

we first show our design rational and how to build a tweak scheduling function of Tweakable TWINE at section 4.3. At section 4.2 we present our proposal Tweakable TWINE. Section 4.4 shows the security evaluation on our proposal. We present hardware implementation results at Section 4.5, and conclude at Section 4.6.

4.2 Specification

Tweakable TWINE is based on TWINE [168], which is a 64-bit block cipher supporting two key lengths of 80 bits and 128 bits. Tweakable TWINE takes a 64-bit tweak value T as an additional input, and consists of a data processing part, a key scheduling function and a tweak scheduling function. The data processing part and the key scheduling function are the same as those of TWINE except additional inputs of tweaks in the data processing part. We, hereafter, refer Tweakable TWINE [168] with an 80-bit key and a 128-bit key to T-TWINE-80 and T-TWINE-128, respectively.

4.2.1 Notation

A bitwise exclusive-OR is denoted by $\oplus$. For binary strings, x and y, $x||y$ denotes their concatenation. Let $|x|$ denote the bit length of x. If $|x| = m$, we may write $x_{(m)}$ to emphasize its bit length. If $|x| = 4c$ for a positive integer c, we write $x \rightarrow (k_0||k_1||\cdots||k_{c-1})$, where $|k_i| = 4$, is the partition operation into the 4-bit sub-blocks. The opposite operation, $(k_0||k_1||\cdots||k_{c-1}) \rightarrow x$, is

similarly defined. The partition operation may be implicit, i.e., we may simply write x_i to denote the i-th 4-bit subsequence for any $4c$-bit string x.

Algorithm $Enc(P_{(64)}, RK_{(32\times36)}, RT_{(24\times36)}, C_{(64)})$

1. $X^1_{0(4)}\|X^1_{1(4)}\|\cdots\|X^1_{15(4)} \leftarrow P$
2. $RK^1_{(32)}\|RK^2_{(32)}\|\cdots\|RK^{36}_{(32)} \leftarrow RK_{(32\times36)}$
3. $RT^1_{(24)}\|RT^2_{(24)}\|\cdots\|RT^{36}_{(24)} \leftarrow RT_{(24\times36)}$
4. **for** $i = 1$ **to** 36 **do**
   $RK^i_{0(4)}\|\cdots\|RK^i_{7(4)} \leftarrow RK^i_{(32)}$, $RT^i_{0(4)}\|\cdots\|RT^i_{5(4)} \leftarrow RT^i_{(24)}$
   $X^i_1 \leftarrow S(X^i_0 \oplus RK^i_0 \oplus RT^i_5) \oplus X^i_1$, $X^i_3 \leftarrow S(X^i_2 \oplus RK^i_1 \oplus RT^i_4) \oplus X^i_3$,
   $X^i_5 \leftarrow S(X^i_4 \oplus RK^i_2) \oplus X^i_5$, $X^i_7 \leftarrow S(X^i_6 \oplus RK^i_3 \oplus RT^i_3) \oplus X^i_7$,
   $X^i_9 \leftarrow S(X^i_8 \oplus RK^i_4 \oplus RT^i_2) \oplus X^i_9$, $X^i_{11} \leftarrow S(X^i_{10} \oplus RK^i_5 \oplus RT^i_1) \oplus X^i_{11}$,
   $X^i_{13} \leftarrow S(X^i_{12} \oplus RK^i_6) \oplus X^i_{13}$, $X^i_{15} \leftarrow S(X^i_{14} \oplus RK^i_7 \oplus RT^i_0) \oplus X^i_{15}$,
   If $i \leq 35$ **for** $h = 0$ **to** 15 **do** $X^{i+1}_{\pi[h]} \leftarrow X^i_h$
5. $C \leftarrow X^{36}_0\|X^{36}_1\|\cdots\|X^{36}_{15}$

Algorithm $Dec(C_{(64)}, RK_{(32\times36)}, RT_{(24\times36)}, P_{(64)})$

1. $X^{36}_{0(4)}\|X^{36}_{1(4)}\|\cdots\|X^{36}_{15(4)} \leftarrow C$
2. $RK^1_{(32)}\|\cdots\|RK^{36}_{(32)} \leftarrow RK_{(32\times36)}$
3. $RT^1_{(24)}\|RT^2_{(24)}\|\cdots\|RT^{36}_{(24)} \leftarrow RT_{(24\times36)}$
4. **for** $i = 36$ **down to** 1 **do**
   $RK^i_{0(4)}\|RK^i_{1(4)}\|\cdots\|RK^i_{7(4)} \leftarrow RK^i_{(32)}$, $RT^i_{0(4)}\|\cdots\|RT^i_{5(4)} \leftarrow RT^i_{(24)}$
   $X^i_1 \leftarrow S(X^i_0 \oplus RK^i_0 \oplus RT^i_5) \oplus X^i_1$, $X^i_3 \leftarrow S(X^i_2 \oplus RK^i_1 \oplus RT^i_4) \oplus X^i_3$,
   $X^i_5 \leftarrow S(X^i_4 \oplus RK^i_2) \oplus X^i_5$, $X^i_7 \leftarrow S(X^i_6 \oplus RK^i_3 \oplus RT^i_3) \oplus X^i_7$,
   $X^i_9 \leftarrow S(X^i_8 \oplus RK^i_4 \oplus RT^i_2) \oplus X^i_9$, $X^i_{11} \leftarrow S(X^i_{10} \oplus RK^i_5 \oplus RT^i_1) \oplus X^i_{11}$,
   $X^i_{13} \leftarrow S(X^i_{12} \oplus RK^i_6) \oplus X^i_{13}$, $X^i_{15} \leftarrow S(X^i_{14} \oplus RK^i_7 \oplus RT^i_0) \oplus X^i_{15}$,
   If $i \geq 2$ **for** $h = 0$ **to** 15 **do** $X^{i-1}_{\pi^{-1}[h]} \leftarrow X^i_h$
5. $P \leftarrow X^1_0\|X^1_1\|\cdots\|X^1_{15}$

Figure 4.2: Encryption and decryption of Tweakable TWINE

4.2.2 Data Processing Part

The data processing part is based on a variant of Type-2 GFS with 16 4-bit nibbles [167]. The round function consists of a 4-bit S-box S, a round-key XOR, a round-tweak XOR and a nibble shuffle operation π, which permutes 16 nibbles as shown in Fig. 4.3. The S-box S and the nibble

shuffle operation π are described in Fig. 4.3. The number of rounds for both of T-TWINE-80 and T-TWINE-128 is 36, where the nibble shuffle operation in the last round is omitted.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	C	0	F	A	2	B	9	5	8	3	D	7	1	E	6	4

h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[h]$	5	0	1	4	7	12	3	8	13	6	9	2	15	10	11	14
$\pi^{-1}[h]$	1	2	11	6	3	0	9	4	7	10	13	14	5	8	15	12

Figure 4.3: S-box S and nibble shuffle π

For $i = 1, \ldots, 36$, i-th round uses a 32-bit (8 nibbles) round key RK^i, which is derived from the secret key $K_{(n)}$ with $n \in \{80, 128\}$, using the key scheduling function, and i-th round also uses a 24-bit (6 nibbles) round tweak RT^i, which is derived from the 64-bit tweak T using the tweak scheduling function. The detailed algorithm of encryption and decryption is given as Fig.4.2.

4.2.3 Tweak Scheduling Function

The tweak scheduling function produces $RT_{(24\times36)}$ from the 64-bit tweak T. It is a permutation-based function as shown in Fig. 4.1. In each round, all nibbles are shifted by 6 nibbles, and 6 nibbles which are inserted to a round function are shuffled using a 6-nibble permutation. The Permutation in the tweak scheduling function π^t is shown in Table 4.1. The detailed algorithm is given as Fig. 4.4.

Algorithm $TweakScheduling(T_{(64)}, RT_{(24\times36)})$

1. $t^1_{0(4)} \| t^1_{1(4)} \| \cdots \| t^1_{16(4)} \leftarrow T$

2. **for** $i = 1$ **to** 36 **do**
   $RT^r_{(24)} \leftarrow t^i_0 \| t^i_1 \| t^i_2 \| t^i_3 \| t^i_4 \| t^i_5 \|$
   for $h = 0$ **to** 15 **do** $t^{i+1}_{\pi^t[h]} \leftarrow t^i_h$

3. $RK_{(24\times36)} \leftarrow RT^1 \| RT^2 \| \cdots \| RT^{36}$

Figure 4.4: Tweak scheduling function of Tweakable TWINE

Table 4.1: Permutation in the tweak scheduling function π^t

h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi^t[h]$	11	10	13	14	12	15	0	1	2	3	4	5	6	7	8	9

4.2.4 Key Scheduling Function

The key schedule produces $RK_{(32\times36)}$ from the secret key, $K_{(n)}$, for $n \in \{80, 128\}$. It is a variant of GFS with few S-boxes, which is the same as one used at the data processing. The 80-bit key schedule uses 6-bit round constants, $CON^i_{(6)} = CON^i_{H(3)} \| CON^i_{L(3)}$ for $i = 1$ to 35.

The 80-bit and 128-bit key schedules are shown in Fig. 4.5. Here, $\mathrm{Rot}\mathbf{z}(x)$ means **z**-bit left cyclic shift of x. We note that CON^i corresponds to z^i in $\mathrm{GF}(2^6)$ with primitive polynomial $z^6 + z + 1$.

4.2.5 Test Vectors

We give test vectors of T-TWINE for each key length. The data are represented in hexadecimal form.

A. T-TWINE-80

Plaintext	: `0123456789abcdef`
Key	: `00112233445566778899`
Tweak	: `fedcba9876543210`
Ciphertext	: `fbb33219433a42f2`

B. T-TWINE-128

Plaintext	: `0123456789abcdef`
Key	: `00112233445566778899aabbccddeeff`
Tweak	: `fedcba9876543210`
Ciphertext	: `ce9e755fffeca2f8`

Algorithm $KeySchedule-80(K_{(80)}, RK_{(32\times36)})$

1. $WK_{0(4)} \| WK_{1(4)} \| \cdots \| WK_{19(4)} \leftarrow K$
2. **for** $r = 1$ **to** 35 **do**
3. $\quad RK^r_{(32)} \leftarrow WK_1 \| WK_3 \| WK_4 \| WK_6 \| WK_{13} \| WK_{14} \| WK_{15} \| WK_{16}$
4. $\quad WK_1 \leftarrow WK_1 \oplus S(WK_0)$
5. $\quad WK_4 \leftarrow WK_4 \oplus S(WK_{16})$
6. $\quad WK_7 \leftarrow WK_7 \oplus 0 \| CON^r_H$
7. $\quad WK_{19} \leftarrow WK_{19} \oplus 0 \| CON^r_L$
8. $\quad WK_0 \| \cdots \| WK_3 \leftarrow \mathrm{Rot4}(WK_0 \| \cdots \| WK_3)$
9. $\quad WK_0 \| \cdots \| WK_{19} \leftarrow \mathrm{Rot16}(WK_0 \| \cdots \| WK_{19})$
10. $RK^{36}_{(32)} \leftarrow WK_1 \| WK_3 \| WK_4 \| WK_6 \| WK_{13} \| WK_{14} \| WK_{15} \| WK_{16}$
11. $RK \leftarrow RK^1 \| RK^2 \| \cdots \| RK^{36}$

Algorithm $KeySchedule-128(K_{(128)}, RK_{(32\times36)})$

1. $WK_{0(4)} \| WK_{1(4)} \| \cdots \| WK_{31(4)} \leftarrow K$
2. **For** $r = 1$ **to** 35 **do**
3. $\quad RK^r_{(32)} \leftarrow WK_2 \| WK_3 \| WK_{12} \| WK_{15} \| WK_{17} \| WK_{18} \| WK_{28} \| WK_{31}$
4. $\quad WK_1 \leftarrow WK_1 \oplus S(WK_0)$
5. $\quad WK_4 \leftarrow WK_4 \oplus S(WK_{16})$
6. $\quad WK_{23} \leftarrow WK_{23} \oplus S(WK_{30})$
7. $\quad WK_7 \leftarrow WK_7 \oplus 0 \| CON^r_H$
8. $\quad WK_{19} \leftarrow WK_{19} \oplus 0 \| CON^r_L$
9. $\quad WK_0 \| \cdots \| WK_3 \leftarrow \mathrm{Rot4}(WK_0 \| \cdots \| WK_3)$
10. $\quad WK_0 \| \cdots \| WK_{31} \leftarrow \mathrm{Rot16}(WK_0 \| \cdots \| WK_{31})$
11. $RK^{36}_{(32)} \leftarrow WK_2 \| WK_3 \| WK_{12} \| WK_{15} \| WK_{17} \| WK_{18} \| WK_{28} \| WK_{31}$
12. $RK_{(32\times36)} \leftarrow RK^1 \| RK^2 \| \cdots \| RK^{36}$

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

Figure 4.5: Key schedules of Tweakable TWINE, for 80-bit and 128-bit keys. S-box S is the same as Fig. 4.2, and key schedule constants, CON^i, are described in the bottom.

4.3 Design Rational

In this section, we describe design goals of Tweakable TWINE, and explain how to design the tweak key scheduling function and choose its parameters. Then we show how to reduce the target candidates by the upper bounds of the minimal number of active S-boxes for each number of inserted tweaks to efficiently search the promising constructions.

4.3.1 Design Goals

Our primal motivation is to build a dedicated TBC on GFN, which is one of the classical structures for block ciphers. To minimize design and evaluation costs, we decide to choose a lightweight block cipher TWINE, which is based on GFN, as an underlying cipher. Then we are able to focus on how to design a tweak scheduling function, and to add it to TWINE. Our design goals of Tweakable TWINE are as follow.

1. Reuse the core of original TWINE.
2. Minimize additional H/W cost, especially area.
3. Keep a nice S/W performance of TWINE by SIMD instructions.
4. Minimize additional rounds to keep throughput of TWINE as possible.

Following the requirement 1, we choose a tweak scheduling function that is independent from the key scheduling function unlike the tweakey framework of SKINNY [27] so that Tweakable TWINE is realized by only adding the new tweak scheduling function to TWINE in software and hardware. Besides, the security in the single-key setting and the related-key setting is reduced to that of TWINE. Thus, we can focus on only the security evaluation in the chosen-tweak setting where the adversary can control of the value of a tweak for Tweakable TWINE because the security in the single-key setting for TWINE is well studied by many researchers.

Following the requirements 2 and 3, we choose a permutation-based tweak scheduling function that outputs d nibble out of 16 nibbles as round tweak nibbles and permutes 16 nibbles in each round as shown in Fig 4.6. While the independence of key and tweak schedule differs from Tweakey framework, our tweak scheduling function itself has a similarity to tweakey scheduling of SKINNY. We further simplify it by removing nibble LFSRs and observe no noticeable security loss. It is well known that the H/W cost (esp. gate size) of the permutation-based scheduling is very small as shuffle layers are implemented by cost-free wire operations in hardware. In software, such shuffle operations are executed by shuffle instructions of SIMD.

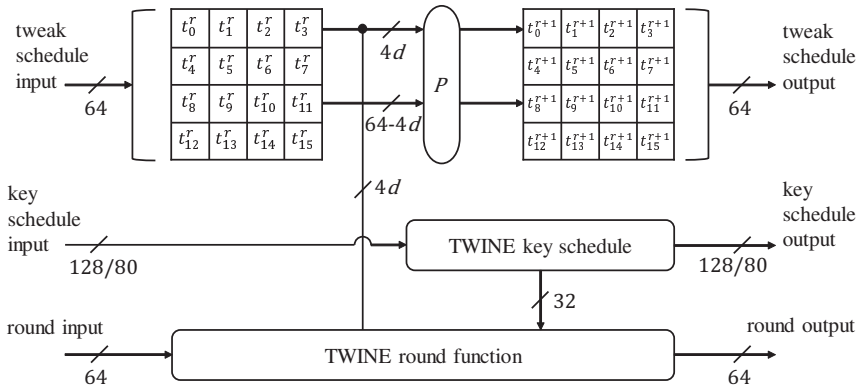


Figure 4.6: Permutaion-based tweak scheduling function

4.3.2 How to Design Permutation-Based Tweak Scheduling Function

To achieve the requirement 4, we properly design the permutation-based scheduling function. Specifically, we need to carefully choose the patterns of the permutation and the locations where tweak nibbles are inserted in the round function, and the number of tweak nibbles to be inputted to each round. As a criteria for finding the best tweak scheduling function, we use *the number of differentially active S-boxes.*

4.3.2.1 Position of Tweak inputs in the Round Function. In each round function, we add d nibble-wise tweak inputs before S-boxes as shown in Fig. 4.8. Thus, the number of candidates of the positions of round tweak inputs is estimated as $\binom{8}{d}$.

4.3.2.2 Permutation. Since the number of possible 16-nibble permutations is $16! \approx 2^{44}$, it is computationally infeasible to evaluate the number of active S-boxes for all permutations. To reduce the search space, we choose the SKINNY-type scheduling function [27] such that in each round, d nibbles which are inserted to a round function are shuffled and all nibbles are shifted by d nibbles as shown in Fig. 4.7, because the SKINNY-type permutation guarantees that each nibble is included in every $\lceil 16/d \rceil$ round. Since the number of position of tweak inputs is 8, the total number of candidates of this-type permutation is estimated as $\sum_{d=1}^{8}(d!)$

An example of $d = 4$ is given as follows. In the figure, we first shuffle the first four nibbles and then shift all 16 nibbles by 4.

$$P_4: \quad (0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15) \rightarrow (4,5,6,7,8,9,10,11,12,13,14,15,3,1,0,2)$$

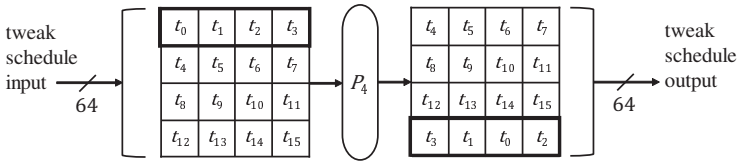


Figure 4.7: Permutation of $d = 4$

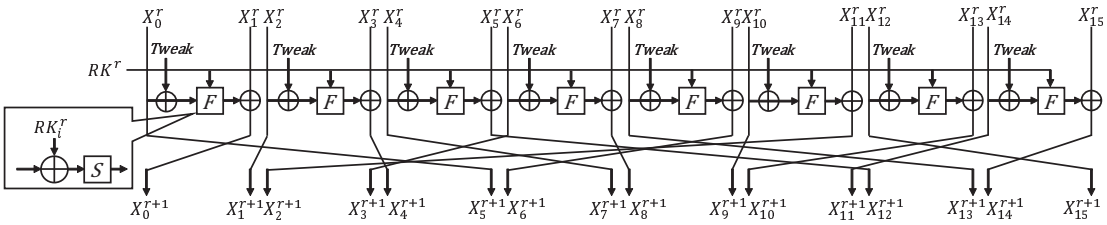


Figure 4.8: Positions for tweak inputs

Therefore, the total number of target tweak scheduling functions to be evaluated is estimated as

$$\sum_{d=1}^{8}(d! \times \binom{8}{d}) \approx 2^{16.74}.$$

For these all candidates, we have evaluated the lower bounds on the number of active S-boxes in each round by the MILP-aided automatic search algorithm (See Section 4.4.1 for details).

4.3.3 Reducing Candidates

Before the evaluation of the minimal number of active S-boxes for all candidates, we show the proposition regarding the upper bound of the minimal number of active S-box for the candidates of each number of the inserted tweaks.

Proposition 2 *For the number of the inserted tweaks is d, the upper bound of the minimal number of active S-boxes over r rounds is $r \times (8 - d)$ in the nibble-wise truncated differential evaluation.*

Proof 1 *Considering the case where the all input tweak differences and plaintext differences are active, all states of differences in the tweak schedule function is active because the tweak schedule function consists of only nibble-wise permutation. In the round function, the S-box in which a tweak is inserted before f function can be inactive, and the S-box in which a tweak is not inserted is always active over all rounds. Therefore, the minimal number of active S-boxes is $(8 - d)$ per 1 round. Thus, for the number of the inserted tweak is d, the upper bound of the minimal number of active S-boxes over r rounds is $r \times (8 - d)$ in the nibble-wise truncated differential evaluation.* □

From Proposition 2, when $d = 8$, those candidates do never achieve 32 active S-boxes. When $d = 7$, those candidates require at least 32 rounds to achieve 32 active S-boxes. From the above, we first evaluated the minimal number of active S-boxes for the candidates for $d = 1$ to 6. As a result, when $d = 6$, we found 32 constructions which achieve 32 active S-boxes over 19 rounds, and it is the smallest round number to achieve 32 active S-boxes in all candidate (see Table 4.2). When $d = 8$ and $d = 7$, there is no candidates which achieve 32 active S-boxes over 19 rounds. Therefore, we do not have to evaluate those candidates, and the candidates that we have to evaluate the number of active S-boxes are reduced to $2^{14.82}$ from $2^{16.74}$.

Among 32 candidates for $d = 6$, we chose the one having more active S-boxes of than the others in each round (1 in Table 4.2).

Table 4.2: The number of active S-box of each round for 32 tweak scheduling functions with $d = 6$ that achieve 32 active Sbox in 19 rounds.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	0	0	0	**2**	**3**	4	**6**	**8**	**10**	12	**15**	**18**	**21**	**23**	**25**	**28**	**30**	**32**	**34**
2	0	0	0	0	1	2	3	4	7	9	11	13	16	20	21	24	27	29	**32**	**34**
3	0	0	0	0	1	2	4	5	6	8	10	12	15	18	22	24	26	28	**32**	33
4	0	0	0	0	1	**3**	4	5	6	7	8	10	12	14	18	23	27	29	**32**	**34**
5	0	0	0	0	1	2	3	5	7	8	9	12	16	17	20	22	24	27	**32**	33
6	0	0	0	0	**2**	2	4	5	6	8	11	**15**	17	20	22	24	26	29	**32**	**34**
7	0	0	0	0	**2**	**3**	4	**6**	**8**	9	11	13	15	18	21	24	26	**30**	**32**	**34**
8	0	0	0	0	1	2	4	**6**	**8**	**10**	12	14	16	19	21	23	26	28	**32**	33
9	0	0	0	0	1	2	3	5	**8**	**10**	**13**	**15**	17	18	22	24	26	29	**32**	33
10	0	0	0	0	1	**3**	4	5	7	9	11	14	17	20	22	23	25	28	**32**	**34**
11	0	0	0	0	**2**	**3**	**5**	**6**	7	8	10	13	16	19	**23**	**25**	27	29	**32**	33
12	0	0	0	0	1	2	3	5	7	**10**	12	**15**	16	19	22	**25**	27	**30**	**32**	33
13	0	0	0	0	1	**3**	4	**6**	7	9	11	14	16	19	22	23	27	29	**32**	33
14	0	0	0	0	1	2	3	5	7	9	12	14	17	20	22	**25**	26	**30**	**32**	33
15	0	0	0	0	**2**	**3**	4	**6**	7	9	11	13	15	18	21	**25**	26	29	**32**	**34**
16	0	0	0	0	**2**	**3**	3	**6**	**8**	**10**	12	14	17	19	21	**25**	**28**	29	**32**	**34**
17	0	0	0	0	1	2	4	5	7	8	10	12	15	17	20	23	26	29	**32**	**34**
18	0	0	0	0	1	2	3	5	6	8	11	14	16	18	21	24	27	29	**32**	**34**
19	0	0	0	0	1	**3**	4	5	6	9	12	**15**	16	19	22	24	27	**30**	**32**	**34**
20	0	0	0	0	1	**3**	**5**	**6**	7	8	10	12	15	18	21	24	26	29	**32**	33
21	0	0	0	0	1	2	3	5	6	8	10	12	15	19	21	24	26	29	**32**	33
22	0	0	0	**1**	1	**3**	4	**6**	**8**	**10**	11	13	15	16	20	24	26	29	**32**	33
23	0	0	0	**1**	1	2	3	5	6	8	11	13	15	19	22	24	26	**30**	**32**	33
24	0	0	0	0	1	2	4	5	7	9	11	13	16	18	21	23	26	29	**32**	33
25	0	0	0	0	1	**3**	4	5	6	8	11	**15**	**18**	**21**	22	**25**	27	29	**32**	**34**
26	0	0	0	0	1	2	3	5	6	9	11	14	17	20	**23**	24	27	29	**32**	33
27	0	0	0	0	1	2	3	5	7	9	12	**15**	17	19	22	24	27	29	**32**	33
28	0	0	0	0	**2**	**3**	4	5	6	9	11	13	16	19	20	23	26	29	**32**	**34**
29	0	0	0	0	1	**3**	4	**6**	**8**	**10**	11	13	16	18	20	23	26	29	**32**	**34**
30	0	0	0	0	**2**	2	2	5	6	7	9	11	15	18	22	**25**	27	**30**	**32**	33
31	0	0	0	0	1	2	3	5	6	7	9	12	16	18	22	24	27	29	**32**	**34**
32	0	0	0	**1**	1	2	3	4	6	8	11	14	15	17	20	23	26	29	**32**	**34**

4.4 Security Evaluation

As mentioned in Section 4.3, the security of Tweakable TWINE in the single-key and related-key settings is reduced to that of TWINE. This section focuses on the security of Tweakable TWINE in the chosen-tweak setting where the adversary fully controls values of a 64-bit tweak. Specifically, we evaluate the security of Tweakable TWINE against differential, linear, impossible differential, and integral attacks by Mixed-Integer Linear Programming (MILP) in the chosen-tweak setting. Our evaluation uses Gurobi Optimizer [97] as an MILP solver.

Tweakable TWINE claims single-key, related-key, and chosen-tweak security and does not claim chosen-tweak-and-related-key security where the adversary can control both of values of tweaks and key relations as it is not relevant in our target application.

4.4.1 Differential/Linear Attack

Differential and linear attacks were proposed by Biham et al. [44] and Matsui [124], respectively. To evaluate the security against differential and linear attacks, we obtain the lower bound on the number of differentially and linearly active S-boxes by the MILP-aided automatic search method, which is proposed by Mouha et al [133]. Since Tweakable TWINE is based on nibble-wise operations, we evaluate all nibble-wise differential and linear trails.

Table 4.3 shows our search results up to 25 rounds in each setting, where AS^D_{SK}, $AS^D_{RK_{80}}$, $AS^D_{RK_{128}}$, AS^D_{CT}, $AS^D_{CTRK_{80}}$ and $AS^D_{CTRK_{128}}$, and AS^L_{CT} are the number of differentially active S-boxes in the single-key setting, the related-key setting (80-bit key), the related-key setting (128-bit key), the chosen-tweak setting, the chosen-tweak-and-related-key setting (80-bit key) and the chosen-tweak-and-related-key settings (128-bit key), respectively, and AS^L_{CT} denotes the number of linearly active S-boxes in the chosen-tweak setting.

Since the maximum differential and linear probability of the S-box is 2^{-2}, 32 active S-boxes ($2^{-2\cdot 32} = 2^{-64}$) are sufficient to guarantee the security against differential and linear attacks. In the chosen-tweak setting, T-TWINE-80 and T-TWINE-128 has at least 32 active S-boxes in 19 rounds. Note that the linear mask in the round function is not canceled by the input linear mask from the tweak schedule. Thus AS^L_{CT} in the chosen tweak setting is the same as the number of active S-boxes in the single-key setting. Therefore, we expect that the full-round Tweakable TWINE has enough immunity against differential and linear attacks in the chosen-tweak setting.

In the chosen-tweak-and-related-key setting, 25 rounds are required to achieve 32 active S-boxes for T-TWINE-80 and T-TWINE-128. Although we do not claim the chosen-tweak-and-related-key security, if it is needed, for the reason that the number of rounds to achieve 32 active S-boxes increase 6 rounds from the chosen-tweak setting, we recommend to add 6 more rounds.

Table 4.3: Lower bound on the number of differentially and linearly active S-boxes in each model

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
AS^D_{SK}	0	1	2	3	4	6	8	11	14	18	22	24	27	30	**32**	-	-	-	-	-	-	-	-	-	-
$AS^D_{RK_{80}}$	0	0	0	0	0	1	3	5	6	8	10	13	16	18	21	23	25	28	**32**	35	-	-	-	-	-
$AS^D_{RK_{128}}$	0	0	0	0	0	1	2	3	4	6	8	10	14	18	20	21	24	26	28	**33**	-	-	-	-	-
AS^D_{CT}	0	0	0	0	2	3	4	6	8	10	12	15	18	21	23	25	28	30	**32**	34	-	-	-	-	-
$AS^D_{CTRK_{80}}$	0	0	0	0	0	0	0	1	3	4	6	9	11	13	15	18	19	21	23	24	25	27	29	30	**32**
$AS^D_{CTRK_{128}}$	0	0	0	0	0	0	1	2	3	5	7	8	10	12	14	15	17	19	21	23	25	27	29	31	**32**
AS^L_{CT}	0	1	2	3	4	6	8	11	14	18	22	24	27	30	**32**	-	-	-	-	-	-	-	-	-	-

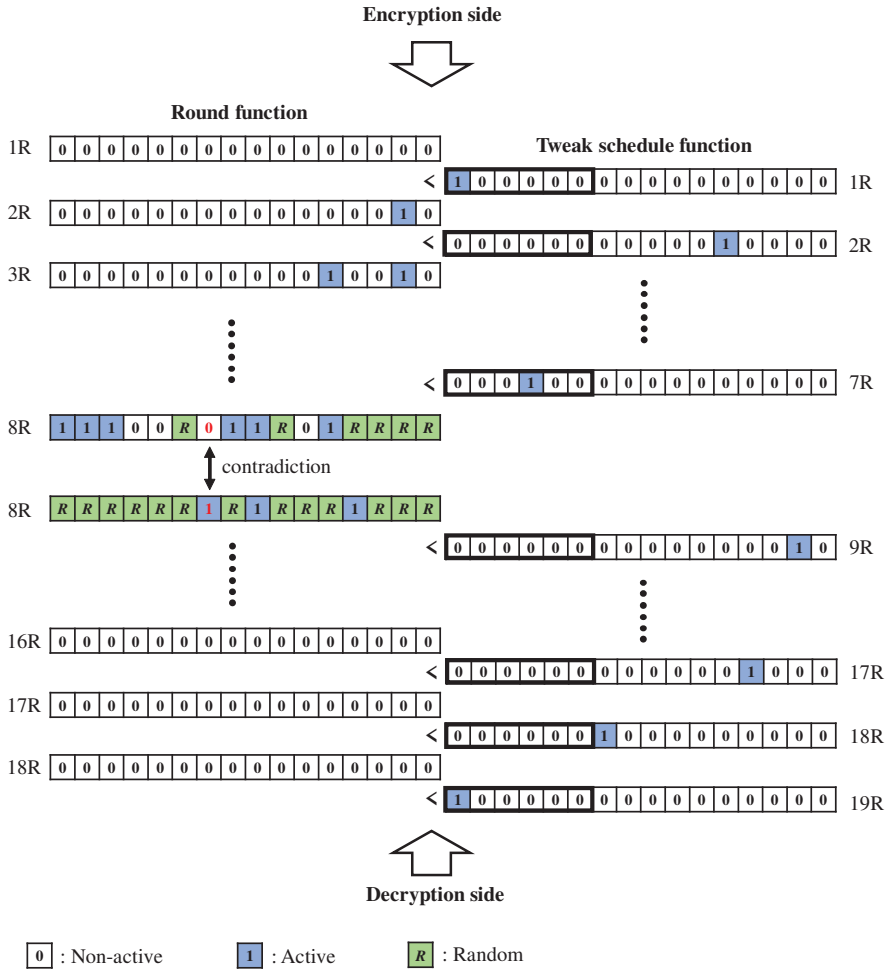


Figure 4.9: 18-round impossible differential characteristic

4.4.2 Impossible Differential Attack

Generally, the cryptanalysis with impossible differentials, which is proposed by Biham et al. [42], is one of the most powerful attacks against Feistel and GFS-based ciphers. By the miss-in-the-middle approach, we search the impossible differential characteristics in which there are up to two active nibbles in the 16 tweak nibbles and 16 plaintext nibbles, and up to two active nibbles

in 16 ciphertext nibbles at the decryption side. Therefore, we explore the following space.

$$\{\binom{16}{1} \times \binom{16}{1}\} + \{(\binom{32}{2} - \binom{16}{2}) \times \binom{16}{2}\} + \{(\binom{32}{2} - \binom{16}{2}) \times \binom{16}{1}\} + \{\binom{16}{1} \times \binom{16}{2}\} \approx 2^{15.70}.$$

As a result, we find an 18-round impossible differential characteristic in the chosen tweak setting as shown in Fig. 4.9. We remark that we do not have to search the input patterns with all active nibble in the 16 plaintext nibbles (and no active nibble in the tweak) because they already have been studied by the original paper of TWINE, showing 64 instances of 14-round characteristic as the longest possible ones among them, in the single-key setting. Due to the structure of Tweakable TWINE, this result is directly applicable to ours. Thus, by controlling tweak inputs, the adversary can improve an impossible differential characteristic by 4 rounds. Since we have 36 rounds, we expect that the full-round Tweakable TWINE has enough immunity against impossible differential attacks.

4.4.3 Integral Attack

The integral attack was first proposed by Daemen et al [63], and then it was formalized by Knudsen and Wagner [110]. After that, it is generalized to the division property by Todo [170], and is defined as follows.

ALL ($\mathcal{A}$) The set contains all possible taken values the same number of times.

CONSTANT ($\mathcal{C}$) All values in the set are equal.

BALANCE ($\mathcal{B}$) The XOR of all values in the set is zero.

UNKNOWN ($\mathcal{U}$) Each value in the set is random.

To evaluate the nibble-based division property, we use an MILP-aided automatic search method which is proposed by Xiang et al. [177], which enable us to efficiently explore the propagation of the division property in the chosen tweak setting. In our evaluation, the division property of tweak is fixed to all $\mathcal{A}$ and division property of plaintext is freely chosen from $2^{16} - 1$ nibble-wise candidates, and the division property of plaintext is fixed to all $\mathcal{C}$ and division property of tweak is freely chosen from 2^{16} nibble-wise candidates. Therefore, the total number of the explored candidates is $2^{17} - 1$.

As a result, we find the 18-round integral distinguisher in the chosen tweak setting as shown in Fig. 4.10. Thus, we expect that the full-round Tweakable TWINE has enough immunity against integral attack in the chosen plaintext setting and chosen tweak setting.

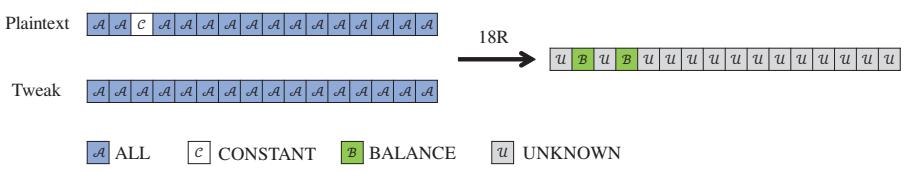


Figure 4.10: 18-round integral characteristic

4.5 Hardware Implementation Results

We evaluated ASIC implementation of Tweakable TWINE and compared it with TWINE. See Table 4.4. We used Yosys ver. 0.7[5] with `osu018_stdcells.lib` process library. This process

[5] `http://www.clifford.at/yosys/`

Table 4.4: Hardware results for round-based, enc-only implementations.

TBC	Yosys (GE)	Known results (GE)	Library used by Known results
T-TWINE-80	2180	–	–
SKINNY-64-128	–	1696 [27]	UMCL18G212T3 standard cell library
CRAFT-128	–	1193 [29]	130nm standard cell library
BC	**Yosys (GE)**	**Known results (GE)**	**Library used by Known results**
TWINE-80	1627	1503 [168]	90nm standard cell library
PRESENT-80	1841	1570 [168]	90nm standard cell library
LED-80	3029	1040 [90]	UMCL18G212T3 standard cell library

library is comparable to TSMC 0.18 um. To see the validity of the tool, we also list some known results. We warn that this consists of several different synthesis environments. Unfortunately, we were not able to compile Tweakable TWINE using tools other than Yosis. We emphasize that, unlike the results of [168], we did not use Scan FF for registers which will reduce total GE counts if available.

The difference between TWINE-80 and T-TWINE-80 are around 550 GEs. Our library has 4 GEs for 1-bit DFF and 3-1 MUX, and 2.3 GEs for 1-bit XOR, thus an implementation of tweak schedule needs 64-bit DFF and 64-bit 3-1 MUX, and 24-bit XOR which amounts to 567 GEs. Thus the difference in size is almost from the additional tweak schedule function.

The results of Table 4.4 imply that T-TWINE-80 requires more GEs than the other tweakable block ciphers listed in the table. However, we focus on how to build a TBC on TWINE while minimizing the additional cost and keeping the security, which is basically a different goal from building a small TBC from scratch (even though TWINE has an excellent hardware performance). Regarding our goal, and that the additional cost is close to be what we can do, we consider that the hardware performance of T-TWINE-80 is reasonably good.

We also evaluated multi-round implementations of T-TWINE-80 as shown in Table 4.5. The fact that Tweakable TWINE has 36 rounds enables a very flexible choice for multi-round implementations with small overhead, which is a property inherited from the original.

4.6 Conclusion

With a motivation of designing a tweakable block cipher based on generalized Feistel structure, we have presented Tweakable TWINE, a tweakable variant of lightweight block cipher TWINE. Our primary design goal is to build a TBC with minimum cost for both design and implementation, so we use TWINE as is and attach an extremely lightweight tweak schedule to it. The design challenge was how to find the best tweak schedule in terms of security and efficiency, and we extensively used Mixed integer linear programming (MILP) solver for this purpose. Consequently, Tweakable TWINE maintains TWINE's efficiency by keeping the same number of rounds, with very little (almost unavoidable) overhead due to the existence of tweak. One of the possible future directions is to apply the same methodology to other block ciphers, and see how efficiently we can turn them into tweakable block ciphers.

Table 4.5: Hardware results for multi-round, enc-only implementation of T-TWINE-80. (Top) ASIC (Yosys with `osu018_stdcells.lib`) (Bottom) FPGA (Intel/Altera `10CL120YF780I7G`, Quartus 18.1)

rnd/clk	Size (GE)	critical path (ps)
1	2180	1077.46
2	2752	1907.98
3	3300	3048.44
4	3793	3717.95
6	4559	5483.03
9	6368	8369.95
12	8795	9947.94
18	20789	13095.40
36	40850	26254.36

rnd/clk	Size (LE)	Fmax (MHz)	critical path (ns)
1	325	343.88	2.9
2	442	244.68	4.1
3	534	158.91	6.3
4	618	124.63	8.0
6	835	85.22	11.7
9	1147	62.50	16.0
12	1449	46.78	21.4
18	2059	31.81	31.4
36	4072	13.12	76.2

5 WARP : Revisiting GFN for Lightweight 128-Bit Block Cipher

In this section, we present WARP, a lightweight 128-bit block cipher with a 128-bit key. It aims at small-footprint circuit in the field of 128-bit block ciphers, possibly for a unified encryption and decryption functionality. The overall structure of WARP is a variant of 32-nibble Type-2 Generalized Feistel Network (GFN), with a permutation over nibbles designed to optimize the security and efficiency. We conduct a thorough security analysis and report comprehensive hardware and software implementation results. Our hardware results show that WARP is the smallest 128-bit block cipher for most of typical hardware implementation strategies. A serialized circuit of WARP achieves around 800 Gate Equivalents (GEs), which is much smaller than previous state-of-the-art implementations of lightweight 128-bit ciphers (they need more than $1,000$ GEs). While our primary metric is hardware size, WARP also enjoys several other features, most notably low energy consumption. This is somewhat surprising, since GFN generally needs more rounds than substitution permutation network (SPN), and thus GFN has been considered to be less advantageous in this regard. We show a multi-round implementation of WARP is quite low-energy. Moreover, WARP also performs well on software: our SIMD implementation is quite competitive to known hardware-oriented 128-bit lightweight ciphers for long input, and even much better for small inputs due to the small number of parallel blocks. On 8-bit microcontrollers, the results of our assembly implementations show that WARP is flexible to achieve various performance characteristics.

5.1 Introduction

5.1.0.1 Lightweight Block Cipher. Due to the increasing need for encryption and authentication on constrained devices, lightweight cryptography has grown to be one of the central topics in symmetric-key cryptography. Among various symmetric-key primitives, the development of lightweight block cipher probably has the longest history. As demonstrated by PRESENT [52], the first generation of lightweight block ciphers, such as KATAN [71], PRINTCIPHER [109] or LED [90], mainly focused on hardware footprint in the standard, round-based constructions. The block size is typically 64 bits or even smaller to reduce the size. Combined with hardware-oriented components (such as a 4-bit S-box and a bit permutation), they achieved a very small hardware footprint compared to the standard AES. Although small-footprint serial AES implementations are possible [15, 132], there is still a gap between what can be done with lightweight block ciphers.

The second generation ciphers aimed at various goals, such as low-latency (PRINCE [53] and QARMA [9]) or low-energy consumption (Midori [11]) or side-channel/fault attack resistance (LS-designs [86], CRAFT [30]), while mostly trying to achieve an equivalent hardware footprint of the first generation ciphers.

5.1.0.2 Importance of 128-bit Cipher. In this section, we focus on lightweight block ciphers with 128-bit block size and 128-bit key. The usefulness of such a primitive is obvious as it can be used as a direct replacement of AES (more precisely `AES-128`), *without changing the mode of operation*. Most of the popular block cipher modes currently used with AES, such as GCM, have birthday bound security, meaning that $O(2^{64})$ input blocks are sufficient to break the scheme. This also implies a certain limitation on 64-bit block ciphers. It is clear that 64-bit block ciphers have been playing the central role in the development of lightweight cryptography. Having said that, birthday attacks with $O(2^{32})$ data complexity can be a real threat[6]. To thwart them, keys

[6]Alternatively, we could use beyond-birthday-bound (BBB) secure modes, however they are generally more complex than the birthday-secure ones, and using complex modes may nullify the merit of using lightweight primitive.

must be renewed very frequently, however this is not trivial in practice (e.g, Sweet32 [39]).

Tweakable block cipher (TBC) of 64-bit block size, such as SKINNY, is another promising way to prevent the birthday attacks of $O(2^{32})$ complexity. It still requires a change of outer modes (though BBB secure modes for TBCs are typically simpler than those for block ciphers) and hence, it generally does not realize a direct replacement of AES.

Consequently, we think lightweight 128-bit block ciphers have their own value. In fact, replacements of AES by lightweight 128-bit ciphers often occur in the development of lightweight authenticated encryption (AE) schemes. For example, COFB [58] and SUNDAE [12] are modern block cipher-based AE modes that were initially specified with AES. Later they were submitted [13, 17] to the ongoing NIST lightweight cryptography project[7] with a 128-bit-block version of GIFT, a family of lightweight block ciphers proposed by Banik et al. [20]. Both submissions [13, 17] are included in the second-round candidates.

As a lightweight replacement of AES, the size of unified encryption and decryption (ED) circuit is important, since some standard/popular block cipher modes, e.g. CBC, OCB [112] and XTS [96], need a block cipher decryption (inverse) circuit as well as an encryption circuit. Besides, when a block cipher is implemented as a co-processor of general-purpose CPUs, we naturally expect the support of both encryption and decryption, as the co-processor is agnostic to the operating modes. Needless to say, an encryption-only circuit is generally smaller and enough for implementing "inverse-free" modes such as CTR or GCM. From these observations, we set our primary goal to build a lightweight 128-bit block cipher that is significantly smaller than prior arts for both encryption-only and unified ED circuits.

5.1.0.3 Our Design. When we look at the current list of lightweight block ciphers, the majorities are Substitution-Permutation Network (SPN) ciphers, such as [27, 52, 53, 90]. However, an SPN is inherently not perfect to our goal, because the decryption circuit generally needs to invert the confusion and diffusion layers. Despite the great research effort on concrete SPN designs using involutory S-boxes and MDS matrices, such as NOEKEON [66], Midori, and QARMA, designing an ultimately lightweight SPN cipher with fully involutory components still seems challenging, when unified ED circuit is a primary target. In particular, if we adopt a serialized datapath, we need recursively defined MDS matrices to be efficient with respect to area [90]. However, it is well known that in fields of characteristic 2, such an MDS matrix can never be involutory [91].

A potential alternative is Generalized Feistel Network (GFN) [140, 180], because it is involutory in nature. The classical Type-2 GFN [180] has been adapted by many ciphers, such as HIGHT [94], CLEFIA [162], and Piccolo [161]. However it has a slow diffusion, which is problematic when the number of sub-blocks (branches) is large. Suzaki and Minematsu [167] (hereafter SM10) proposed a way to greatly improve the diffusion of GFN by just changing the permutation of branches from the rotation originally used by Type-2 GFN. They also showed r-branch permutations achieving the fastest diffusion up to $r = 16$. Indeed, TWINE [168] and LBlock [176] are 64-bit block, 16-branch GFN ciphers that can be seen as concrete instantiations of SM10. It is interesting to note that, GFN ciphers of larger-than-16 branches have been actively studied from the viewpoint of permutation design (see below), however no concrete, purely GFN-based block ciphers have been proposed, to the best of our knowledge[8]. In this section, we revisit GFN to investigate if it fulfills our needs. Specifically, we extend the idea of SM10 to build a 128-bit, 32-branch (nibble) GFN cipher with 128-bit key, named WARP[9]. As observed by SM10, one can achieve the diffusion round (the number of rounds needed for diffusing any input differ-

[7] `https://csrc.nist.gov/Projects/lightweight-cryptography`

[8] Liliput [33] is a 128-bit TBC built on a variant of GFN (EGFN [34]). It has a different linear layer structure from GFN and has 16 branches.

[9] The name comes from the resemblance of the cipher structure to strings in a loom.

ence to the whole output) as low as $2\log_2 r$, which implies that a good 128-bit, 32-nibble GFN cipher may only need two more rounds from the case of 64-bit, 16-nibble GFN ciphers. The big challenge is to determine a 32-branch permutation. The diffusion property of r-branch permutations for $r > 16$ has been recently studied, and made a significant progress since SM10 [55, 73]. However, these studies do not give a direct answer to us, as we need a permutation having not only a fast full diffusion but also a high immunity against known attacks (differential/linear/impossible differential/integral/division etc). Because an exhaustive search over all 32-branch permutations is computationally infeasible, we define a subset of permutations that are suitable to serial circuits and search over it with an Mixed Integer Linear Programming (MILP) solver, based on the development of MILP-aided security evaluation initiated by Mouha et al. [133]. Notably, we found that the 32-branch permutations with 9-round full diffusion (which is 1 round smaller than what SM10 showed) by Derbez et al. [73] are not suitable because the number of active S-box grows very slowly. Our permutation has 10-round full diffusion, however performs much better in terms of the number of active S-boxes (see Sect. 5.3.1).

We adopt an S-box of Midori for its small delay and area. It is also very efficient for threshold implementations which is very important when side-channel attacks are possible.

The key schedule of WARP is ultimately simple: the 128-bit key is divided into two 64-bit halves and they are alternately used, i.e. the parity of the round number determines which half is used. This removes a need of additional register. Such permutation-based key scheduling schemes have been employed by a number of recent block ciphers, e.g, LED [90], Piccolo [161] and CRAFT [30] as well as stream ciphers [19, 128]. In addition, every sub-key is XORed after S-box is applied to avoid the complement property of Feistel-Type Structures [50], following the idea of Piccolo [161].

5.1.0.4 Implementation Results. Combining these components, we achieved 763 GE for the bit-serial encryption-only circuit, which is, to our knowledge, the lowest number of 128-bit block cipher hardware implementation to date. Moreover, due to the low-energy and low-delay S-box, the 2-round unrolled implementation of WARP achieved significantly better energy consumption as compared to Midori, which is the current state-of-the-art design as a 128-bit low-energy cipher. For the unrolled (Enc-only) implementations, WARP is smaller than QARMA, while keeping relatively small delay, around 1.6 of $\texttt{QARMA-128}_{11}$. We also conducted threshold implementations of WARP for protection against first-order side-channel attacks. The results are quite impressive (Table 5.10 at Sect. 5.5.4). All in all, WARP has pretty good performance for multiple hardware metrics not only in size.

For software metrics on microcontrollers, the design of WARP makes it flexible to make different trade-offs. We report performance characteristics of our assembly implementations on 8-bit AVR following various methods. The results show that, for WARP, it is possible to achieve competitively small code size and extremely low RAM consumption, with acceptable execution time.

Finally, thanks to the software-friendly structure of GFN, we report a very efficient software implementation of WARP on modern high-end CPUs equipped with SIMD instructions. Unlike known bitslice implementation of recent lightweight ciphers, which need many block to be processed in parallel, we use a vector permutation (`vperm`) instruction, in a similar manner to TWINE [32]. This allows us to work with small (or no) parallelism. Surprisingly, the results on modern Intel processors are very competitive to the bitslice implementations of several state-of-the-art lightweight ciphers (GIFT, SKINNY and SIMON [24]). This gives another advantage to WARP when the operating mode is serial, say CBC-MAC or lightweight, serial authenticated encryption mode such as CLOC [101], SAEB [134], or COFB [58].

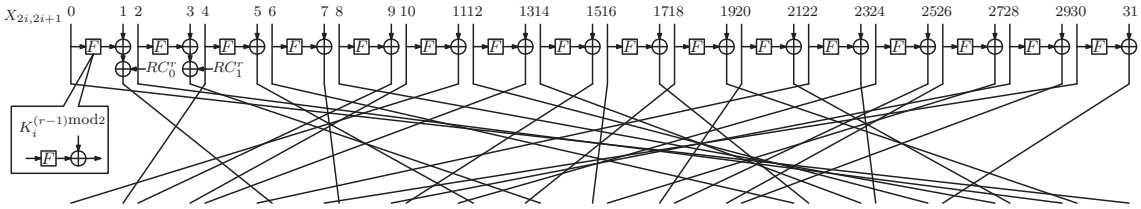


Figure 5.1: Round Function of WARP.

Table 5.1: 4-bit S-box S.

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

5.1.0.5 Organization. This section is organized as follows. We first present the specification of our cipher at Section 5.2. We provide our design rationale, such as 32-branch permutation and S-box, at Section 5.3. Section 5.4 describes the details of security evaluations against major cryptanalysis methods. Section 5.5 and Section 5.6 provide our hardware and software implementations. Finally, we conclude at Section 5.8.

5.2 Specification

WARP is a 128-bit block cipher with a 128-bit key. The general structure of WARP is a variant of the 32-branch Type-2 GFN. A 128-bit plaintext M and a ciphertext C are loaded into a 128-bit internal state in encryption and decryption processes, respectively. The internal state is expressed as 32 nibbles, $X = X_0 \parallel X_1 \parallel \ldots \parallel X_{31}$, where $X_i \in \{0,1\}^4$. A 128-bit secret key K is denoted as two 64-bit keys K^0 and K^1, i.e. $K = K^0 || K^1$, where $K^i \in \{0,1\}^{64}$. K^0 and K^1 are also expressed as 16 nibbles, $K^0 = K_0^0 \parallel K_1^0 \parallel \ldots \parallel K_{15}^0$, where $K_i^0 \in \{0,1\}^4$, and $K^1 = K_0^1 \parallel K_1^1 \parallel \ldots \parallel K_{15}^1$, where $K_i^1 \in \{0,1\}^4$, respectively.

5.2.0.1 Round Function. The round function of WARP consists of a 4-bit S-box $S : \{0,1\}^4 \to \{0,1\}^4$, a nibble XOR : $\{0,1\}^4 \times \{0,1\}^4 \to \{0,1\}^4$, and a shuffle operation $\pi : \{0,\ldots,31\} \to \{0,\ldots,31\}$ applied to 32 nibbles. The round function applies a non-linear unit transformation involving a single S evaluation and round-key addition for each of two consecutive nibbles, adds a round constant, and applies π to all 32 nibbles. See Fig. 5.1. The S-box S is described in Table 5.1. The shuffle π and its inverse π^{-1} are described in Table 5.2.

5.2.0.2 Encryption and Decryption. The number of rounds of WARP is 41, where the nibble shuffle operation π in the last round is omitted. For $i = 1,\ldots,41$, the i-th round uses a

Table 5.2: Shuffle π on 32 nibbles.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(x)$	31	6	29	14	1	12	21	8	27	2	3	0	25	4	23	10
$\pi^{-1}(x)$	11	4	9	10	13	22	1	30	7	28	15	24	5	18	3	16
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\pi(x)$	15	22	13	30	17	28	5	24	11	18	19	16	9	20	7	26
$\pi^{-1}(x)$	27	20	25	26	29	6	17	14	23	12	31	8	21	2	19	0

Table 5.3: Round constants (listed in hexadecimal).

r		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
RC_0^r		0	0	1	3	7	f	f	f	e	d	a	5	a	5	b	6	c	9	3	6
RC_1^r		4	c	c	c	c	c	8	4	8	4	8	4	c	8	0	4	c	8	4	c
r	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
RC_0^r	d	b	7	e	d	b	6	d	a	4	9	2	4	9	3	7	e	c	8	1	2
RC_1^r	c	8	4	c	8	4	8	0	4	8	0	4	c	c	8	0	0	4	8	4	c

64-bit (16 nibbles) round key RK^i. Then, an i-th round key RK^i is given as $RK^i = K^{(i-1) \bmod 2}$.

The encryption algorithm of WARP is given in Fig. 5.2. The decryption algorithm is omitted here. It is obtained by just changing π to its inverse π^{-1}.

WARP uses LFSR-based round constants. A state of 6-bit LFSR is written as $(\ell_5, \ell_4, \ell_3, \ell_2, \ell_1, \ell_0)$ and is initialized to 000001. It is updated in each round as

$$(\ell_5, \ell_4, \ell_3, \ell_2, \ell_1, \ell_0) \leftarrow (\ell_4, \ell_3, \ell_2, \ell_1, \ell_0, \ell_0 \oplus \ell_5).$$

Using this LFSR, we define two nibbles $RC_0 = (\ell_5, \ell_4, \ell_3, \ell_2)$ and $RC_1 = (\ell_1, \ell_0, 0, 0)$. RC_0 and RC_1 are xored to the first and third nibbles of the state (note that the numbering of the nibbles is from 0 to 31) after the $X_{2i+1} \leftarrow S(X_{2i}) \oplus K_i^{(r-1) \bmod 2} \oplus X_{2i+1}$ operation. Let RC_0^r and RC_1^r be the r-th round constants. For completeness, we list (RC_0^r, RC_1^r) for all $r = 1, \ldots, 41$ in Table 5.3.

Algorithm Encryption(K, M)

1. $(K_0^0 \| K_1^0 \| \ldots \| K_{15}^0, K_0^1 \| K_1^1 \| \ldots \| K_{15}^1) \leftarrow K$
2. $X_0 \| X_1 \| \ldots \| X_{31} \leftarrow M$
3. **for** $r = 1$ **to** 40 **do**
4. **for** $i = 0$ **to** 15 **do**
5. $X_{2i+1} \leftarrow S(X_{2i}) \oplus K_i^{(r-1) \bmod 2} \oplus X_{2i+1}$
6. **end for**
7. $X_1 \leftarrow X_1 \oplus RC_0^r$, $X_3 \leftarrow X_3 \oplus RC_1^r$
8. $X'_0 \| X'_1 \| \ldots \| X'_{31} \leftarrow X_0 \| X_1 \| \ldots \| X_{31}$
9. **for** $i = 0$ **to** 31 **do**
10. $X_{\pi[j]} \leftarrow X'_j$
11. **end for**
12. **end for**
13. **for** $i = 0$ **to** 15 **do**
14. $X_{2i+1} \leftarrow S(X_{2i}) \oplus K_i^0 \oplus X_{2i+1}$
15. **end for**
16. $X_1 \leftarrow X_1 \oplus RC_0^{41}$, $X_3 \leftarrow X_3 \oplus RC_1^{41}$
17. $C \leftarrow X_0 \| X_1 \| \ldots \| X_{31}$
18. **return** C

Figure 5.2: Encryption algorithm of WARP.

5.2.0.3 Claimed Security WARP claims single-key security, and does not claim any security in related-key and known/chosen-key settings.

5.2.0.4 Test Vector

Table 5.4 shows the test vectors of WARP.

Table 5.4: Test vectors.

B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
K	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
M	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
C	2	4	C	E	0	A	8	E	F	D	9	F	3	2	D	E	5	2	9	D	5	F	D	F	4	5	7	0	3	A	8	D
K	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
M	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9	9	A	A	B	B	C	C	D	D	E	E	F	F
C	9	2	3	C	6	4	F	9	2	8	2	7	E	E	6	2	B	9	6	6	7	D	D	2	5	4	8	F	B	1	2	C
K	0	A	C	D	0	2	2	F	6	8	0	A	5	4	7	F	E	E	0	3	C	0	8	6	7	B	0	9	E	3	D	7
M	A	F	6	C	D	D	9	0	F	C	5	A	6	E	A	A	8	9	7	B	C	D	1	2	0	8	D	3	9	1	E	1
C	6	1	2	3	9	9	5	F	1	9	2	4	D	3	1	4	2	5	6	4	1	A	C	D	D	0	5	8	D	D	4	6

B: Branch Index K: Master key M: Plaintext C: Ciphertext

5.3 Design Rationale

As described, the goal of WARP is a 128-bit block cipher enabling small hardware implementation, both for encryption-only and unified ED circuits, and both for round-based and serial architectures. We detail the rational of our design choice for each component of GFN below.

5.3.1 Branch Size and Permutation

We choose to use 32-nibble GFN with a 4-bit S-box, instead of 16-byte GFN with an 8-bit S-box. Although the latter option allows to reuse most of the known design/cryptanalytic results on 16-branch GFN (SM10, TWINE or LBlock and their cryptanalysis such as [49]), 8-bit S-box is much inferior to 4-bit S-box in terms of size/delay/energy.

We need a $r = 32$-branch permutation that is good in terms of diffusion round and resistance to the major attacks, such as differential and linear attacks. Despite the recent research on many-branch GFN [34,55,167], this remains a hard problem, simply because the number of permutation quickly grows ($r!$). When $r = 2^s$, SM10 shows an r-branch permutation of diffusion round being $2s$ based on de Bruijin graph, however, according to our random search, there is a huge number of 32-branch permutations having diffusion round of $2\log_2 32 = 10$. Besides, the differential/linear Active S-box (AS-box) counts are very different among them, which suggests that we need another criteria before searching.

After some experiments, we limit ourselves to permutations allowing efficient serial hardware implementations, which is our main focus (See Section 5.5 for hardware implementation). In more detail, we searched all permutations of LBlock-like structure that consists of one 16-branch permutation composed of two identical 8-branch permutations, and one rotation on 16 branches with an amount of rotation from 0 to 15 nibbles as shown in Fig. 5.3. The resulting search space has size $8! \times 16 \approx 2^{19.3}$. The search over this space found 152 candidates of diffusion round 10. We conducted MILP-based differential AS-box counting for them. This evaluation requires about 2 days on computer equipped with 44 cores and 64 GB RAM.s Among them, 21 candidates achieved AS-box of ≥ 64 (which is needed for security) at 19 rounds (and no candidates achieved it at 18 rounds), and 8 out of 21 achieved AS-box of 66, which was the largest among them. These 8 permutations are not isomorphic, however as far as we investigated, the attack characteristics for other attacks (linear AS-box, impossible differential characteristics etc) are identical for all of them.

Our investigation implies that they are equivalently secure in practice. Moreover, there is no difference from the implementation aspects too. Thus, we arbitrarily chose one among them. A LBlock-like equivalent round function of WARP is shown in Figure 5.4.

Table 5.5: Four equivalent classes of 32-branch permutations with 9-round full diffusion [73].

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi'_0(x)$	3	4	5	12	7	24	9	20	11	2	1	26	15	8	17	30
$\pi'_1(x)$	3	10	5	12	7	26	9	20	11	8	1	24	15	18	17	30
$\pi'_2(x)$	3	4	5	2	7	24	9	16	11	14	1	28	15	10	17	8
$\pi'_3(x)$	3	14	5	12	7	28	9	18	11	22	1	30	15	2	17	24
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\pi'_0(x)$	19	14	21	18	23	28	13	10	27	16	29	6	25	22	31	0
$\pi'_1(x)$	19	4	21	2	23	28	13	14	27	22	29	6	25	16	31	0
$\pi'_2(x)$	19	26	21	22	23	20	13	30	27	18	25	6	31	12	29	0
$\pi'_3(x)$	13	4	21	26	23	10	19	8	27	20	25	16	29	6	31	0

Recently, Derbez et. al [73] showed four equivalent classes of 32-branch permutations achieving full diffusion after 9 rounds, while WARP requires 10 rounds. However, our MILP-based evaluation revealed that the number of active S-boxes of these grows much slower than ours. Indeed, these require at least 32 rounds for achieving AS-box of ≥ 64. Since WARP achieves it with only 19 rounds, the permutation of WARP is better than them as a 32-branch permutation. Table 5.5 shows four equivalent classes of 32-branch permutations of $\pi'_0(x)$, $\pi'_1(x)$, $\pi'_2(x)$, and $\pi'_3(x)$ achieving 9-round full diffusion found by Derbez et. al [73]. Table 5.6 is a comparison of lower bounds on the number of active S-boxes for WARP and four permutations by our MILP-based Active S-boxes counting. As shown in Table 5.6, the number of active S-boxes of $\pi'_0(x)$, $\pi'_1(x)$, $\pi'_2(x)$, and $\pi'_3(x)$ grows much slower than WARP. Specifically, $\pi'_0(x)$, $\pi'_1(x)$ and $\pi'_2(x)$, $\pi'_3(x)$ require at least 32 and 48 rounds for achieving AS-box of ≥ 64, respectively.

Table 5.6: Lower bounds on the number of Active S-boxes for WARP and four permutations of $\pi'_0(x)$, $\pi'_1(x)$, $\pi'_2(x)$, and $\pi'_3(x)$

# of rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
WARP	0	1	2	3	4	6	8	11	14	17	22	28	34	40	47	52	57	61	**66**
$\pi'_0(x)$ [73]	0	1	2	3	4	6	8	11	14	19	22	24	26	28	30	32	34	36	38
$\pi'_1(x)$ [73]	0	1	2	3	4	6	8	11	14	19	22	24	26	28	30	32	34	36	38
$\pi'_2(x)$ [73]	0	1	2	3	4	6	8	10	12	12	14	16	16	18	20	20	22	24	24
$\pi'_3(x)$ [73]	0	1	2	3	4	6	8	10	12	12	14	16	16	18	20	20	22	24	24

5.3.2 S-box

According to [11], the small path delay and the small gate area lead to low-energy implementation. We searched a small-delay and lightweight 4-bit S-box which fulfills the following requirements: (1) the maximal probability of a differential is 2^{-2}, (2) the maximal absolute bias of a linear approximation is 2^{-2} and (3) preferably belonging one of the 30 cubic classes (as given in [46]) that allows decomposition into two quadratic s-boxes, so that it can be used to implement a 1st order threshold implementation with 3 shares. This helps us have a very lightweight threshold circuit as well. As a result, we decide to use S-box of Midori (Sb_0). Note that other S-boxes used in low-latency ciphers such as PRINCE and QARMA do not satisfy the requirement (3).

5.3.3 Key Schedule

The key schedule uses alternately the upper and lower half of the 128-bit key in alternate rounds. This requires only a multiplexer to filter appropriate portions of the round key in each round. As already outlined in [11,29], an elaborate key schedule function requires a register element to store and update the key, which is costly in terms of area and energy consumption. Moreover, a simple

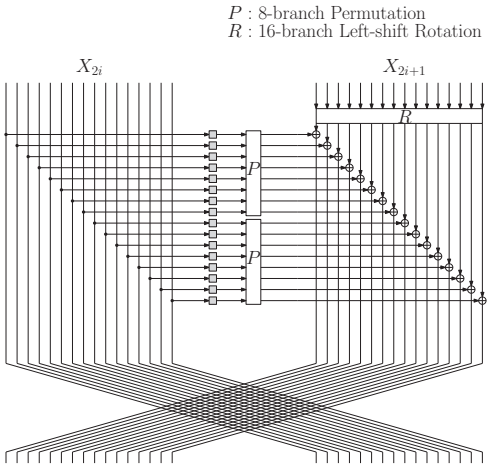


Figure 5.3: General LBlock-like round function.

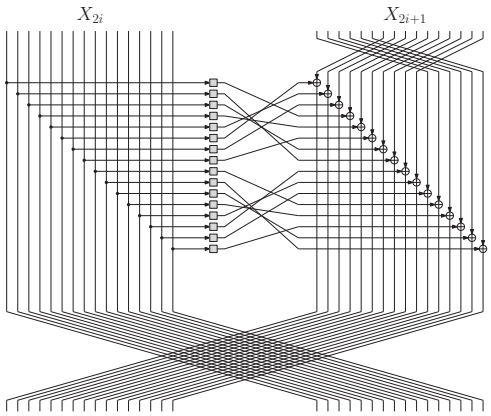


Figure 5.4: Equivalent round function of WARP in LBlock-like structure.

key schedule is particularly beneficial to unified ED circuits, because additional hardware is not required to construct an inverse key schedule function. A key alternating cipher like WARP with odd number of rounds, uses the same upper half of the key in the first and the last encryption round (and indeed in all odd rounds) which implies that the decryption routine would also use the upper half of the key in the first, last and all odd rounds. Thus, the order of upper/lower half of keys used in successive rounds is exactly the same for encryption and decryption, thus no additional overhead is imposed to implement decryption alongside the encryption in hardware. In addition, the key XOR operation is applied after the S-box to avoid the complement property of Feistel-Type Structures [50], following the idea of Piccolo [161].

5.3.4 Round Constants

We use LFSR-based round constants as it is simple and efficient to implement in hardware. We use 6-bit LFSR with a primitive connection polynomial, which has a period of 63, and hence sufficient to cover 41 rounds used in WARP.

5.4 Security Evaluation

We evaluate the security of WARP against differential, linear, integral, impossible differential, invariant and meet-in-the-middle attacks. Among them, the 21-round impossible differential attack is considered to be the most efficient for WARP. In our evaluation, we do not expect an effective key-recovery attack on up to 32 rounds of WARP by using this 21-round impossible differential distinguisher or even using other ones. Consequently, we conclude that the full-round of WARP is expected to be resistant to those attacks. The detailed evaluation results will be given as follows.

5.4.1 Differential/Linear Attack

Differential cryptanalsis [45] and linear cryptanalsis [124] are among the most powerful techniques available for block ciphers. To evaluate the security against differential and linear attacks, we compute the lower bound for the number of differentially and linearly active S-boxes with a MILP-aided automatic search method, which was proposed by Mouha et al. [133]. We use Gurobi [97] as the solver and search for all nibble-wise truncated differential and linear characteristics.

Table 5.7 shows the minimum number of differentially and linearly active S-boxes for up to 19 rounds in the single-key setting, where AS_D and AS_L denote the number of differentially and linearly active S-boxes, respectively. It can be observed from Table 5.7 that WARP has more than 64 active S-boxes after 19 rounds. Since the maximum differential probability and absolute linear bias of the S-box of WARP are both 2^{-2} and the nibble-wise full diffusion requires 10 rounds, even with a 19-round differential distinguisher, we expect that an effective key-recovery attack cannot reach up to $19 + 12 = 31$ rounds. In a word, the full-round WARP is secure against differential and linear attacks.

Table 5.7: The lower bound for the number of differentially and linearly active S-boxes in the single-key setting.

#Rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
AS_D/AS_L	0	1	2	3	4	6	8	11	14	17	22	28	34	40	47	52	57	61	**66**

5.4.2 Impossible Differential Attack

Generally, an impossible differential attack [42] is one of the most powerful attacks against Feistel-type ciphers. The impossible differential attack exploits a pair of input-output difference denoted by Δ_{in} and Δ_{out} such that Δ_{in} will never reach Δ_{out} after several rounds.

As mentioned in Section 5.3, WARP achieves the full diffusion after 10 rounds at both the encryption and decryption sides in nibble-wise. Based on a more detailed investigation, we found that the full diffusion requires 12 rounds at both the encryption and decryption sides in bit-wise. Hence, there should be no probability-1 bit-wise impossible differential over 24 rounds.

In order to obtain the longest impossible differential distinguisher, we utilize an impossible differential search tool based on MILP designed by Sasaki and Todo [158]. Specifically, we evaluate the search space such that the plaintext difference and ciphertext difference activate only one bit, respectively. To model the propagation of differences through the 4-bit S-box, we take into account the differential distribution table for the 4-bit S-box. Based on the method as proposed in [166], it can be modeled with the linear inequalities.

As a result, we find the following 21-round impossible differential distinguisher.

$$
\begin{aligned}
&(\texttt{0000000000001000000000000000000000000000000000000000000000000000}\\
&\texttt{0000000000000000000000000000000000000000000000000000000000000000})\\
\xrightarrow{\text{21 rounds}}\ &(\texttt{0000000000000000000000000000000000000000000010000000000000000000}\\
&\texttt{0000000000000000000000000000000000000000000000000000000000000000})
\end{aligned}
$$

According to Boura et al's work [54] in ASIACRYPT 2014 and the corresponding interpretation [72] by Derbez in FSE 2016, when extending the 21-round impossible differential distinguisher for 10 rounds, the required time complexity of the key-recovery attack is almost close to a pure exhaustive key search. Therefore, we do not expect an effective key-recovery attack on up to 32 rounds of WARP by using this 21-round impossible differential distinguisher. In a word, we expect that the full-round WARP is secure against the impossible differential attack.

5.4.3 Integral Attack

The integral attack was first proposed by Daemen et al. [63] and it was later formalized to the integral property by Knudsen and Wagner [110]. We define the four states for a set of 2^n n-bit cell: **A**: if $\forall i, j\ i \neq j \Leftrightarrow x_i \neq x_j$, **C**: if $\forall i, j\ i \neq j \Leftrightarrow x_i = x_j$, **B**: $\bigoplus_i^{2^n-1} x_i$, and **U**: Other. The

integral attack was further generalized to the division property by Todo [170], which can exploit the hidden feature between **A** and **B** states.

To evaluate the nibble-based division property, we use a MILP-aided automatic search method proposed by Xiang et al. [177], which enables us to efficiently explore the propagation of the division property. Specifically, we evaluate all the cases where 1, 2, 3 nibbles out of 32 nibbles are **C** and the others are all **A** in plaintexts. Thus, we need to evaluate $2^{23.2}\left(=\binom{32}{1}+\binom{32}{2}+\binom{32}{3}\right)$ nibble-wise patterns.

In this way, we find the following 20-round integral distinguisher.

$$(\mathbf{AAAAAACAAAAAAAAAAAAAAAAAAAAAAAA})$$
$$\xrightarrow{20\text{ rounds}}(\mathbf{UUUBUBUUUUUUUUBUUUUUUUUUUUUBUU})$$

However, due to the high data complexity of this integral distinguisher, one can extend only 1 round to achieve a key-recovery attack and its time complexity is almost close to an exhaustive key search. One may find an integral distinguisher by using a lower data complexity, but the number of rounds will be reduced. Considering that the nibble-wise full diffusion requires 10 rounds and that a common integral distinguisher covering larger rounds always requires a higher data complexity, we expect that the full-round WARP is secure against integral attacks.

5.4.4 Meet-in-the Middle Attack

We evaluate the security against the meet-in-the-middle attack following the method appeared in the self-evaluation of Midori [11] and CRAFT [30]. The 10-round full diffusion property guarantees that any inserted key-bit non-linearly affects all branches after the 10 rounds in the forward and the backward directions, respectively. Thus, the possible number of rounds used for the partial matching (PM) [157] is estimated as 19 $(=(10-1)+(10-1)+1)$. The condition for the initial structure (IS) [157] is that key differential trails in the forward direction and those in the backward direction do not share active non-linear components. For WARP, since any key differential affects all 16 S-boxes after at least 10 rounds in the forward and the backward directions, there is no such differential which shares active S-box in more than 10 rounds. Thus, the number of rounds used for IS is upper bounded by 9. Assuming that the splice-and-cut technique allows an attacker to add more 3 rounds in the worst case, at most 32-round (19 + 10 + 3) MitM attack may be feasible. However, because of the iterated key insertions of K_0 and K_1 for every two round, we consider that it is difficult to mount a 32-round attack on WARP.

5.4.5 Invariant Subspace Attack

We use LFSR-based round constants in each round. Following the notions presented in [25], we first tried to find the smallest L-invariant subspace that contains all roundkey differences. Here, L denotes the transformation that describes the linear layer in WARP. Since WARP adds two key halves in an alternating fashion, the master keys repeat every other round, the set of roundkey differences in the even/odd rounds are given by

$$D_{even}=\{\mathbf{RC}^r\oplus\mathbf{RC}^{r+2}:i\in\{0,2,\ldots\}\},\ D_{odd}=\{\mathbf{RC}^r\oplus\mathbf{RC}^{r+2}:i\in\{1,3,\ldots\}\},$$

where $\mathbf{RC}^r$ is the 128-bit vector defined as $(0^4\parallel RC_0^r\parallel 0^4\parallel RC_1^r\parallel 0^{112})$. Denoting $D=D_{even}\cup D_{odd}$, we try to find $W_L(D)$, which denotes the smallest L-invariant subspace containing D. We found that $W_L(D)$ is a subspace of dimension 124, which does not automatically guarantee resistance to subspace attacks. As the invariant attack applies only if there is a non-trivial invariant g for the S-box layer such that $W_L(D)\subset LS(g)$, where $LS(g)$ is the subspace of all linear structures of the function g. We ran Algorithm 1 of [25] on $Z=W_L(D)$ first to see if $S(Z)$ hits all

the cosets of Z. Experimentally we found that it does indeed, leading us to conclude that g is the constant function which guarantees security against subspace attacks.

5.5 Hardware Performance

One of the principal objectives for our design was efficiency in constrained platforms with respect to multiple metrics of lightweight cryptography. Hence we looked at area, energy and latency which are widely acknowledged to be factors that determine the quality of a design. We first convert the round function to an LBlock-type architecture that helps us construct an efficient serial hardware architecture for WARP. Consider a 2-branch Feistel network, with a 128-bit block composed of $X_a||X_b$ (each of 64 bits). Further let $X_a[i], X_b[i], K[i], \forall\ i \in [0, 15]$ denote the individual nibbles of the branches, and the roundkey respectively. Then the LBlock-type function defined below in Fig 5.5, can also be used to define the specifications of WARP.

Round Function(X_a, X_b, K)

for $i = 0$ to 15 **do**
  $T[\pi[i]] \leftarrow S(X_a[i]) \oplus K[\pi[i]]$, `-- Sbox, shuffle left branch,addkey`
  $U[i] \leftarrow X_b[6 + i \bmod 16]$, `-- Rotate 6 nibbles (right branch)`
for $i = 0$ to 15 **do**
  $U[i] \leftarrow U[i] \oplus T[i]$, `-- Add left, right branches`
$U[0] \leftarrow U[0] \oplus RC_0$, $U[1] \leftarrow U[1] \oplus RC_1$ `-- Round const add`
for $i = 0$ to 15 **do**
  $X_b[i] \leftarrow X_a[i]$, $X_a[i] \leftarrow U[i]$ `-- Swap left, right branches`

Figure 5.5: Alternative definition of Round Function

In this definition, π is a permutation which maps i to the i-th element of the following set: $\{3, 7, 6, 4, 1, 0, 2, 5, 11, 15, 14, 12, 9, 8, 10, 13\}$. It is elementary to show that the encryption routine defined by 41 iterations of the round function in Fig 5.5 (with the left-right swap omitted in the last round) is equivalent to the definition of the encryption algorithm of WARP up to a shuffle of nibbles.

5.5.1 Nibble Serial Architecture

Figure 5.6 shows the architecture for WARP. Each storage element colored yellow/white in the figure is a 4-bit scan/normal flip-flop respectively. Apart from 4-bit xor gates required for round key addition, left-right branch addition, and round-constant addition, we have used a few multiplexers to manoeuvre data through the circuit. The circuit uses only one S-box, and in addition we have a key-multiplexer that filters roundkey nibbles from the 128-bit master key (which is not shown in the figure for space constraints). The circuit computes one round function of WARP in 48 cycles. Following is the cycle-by-cycle description of the circuit operations.

Cycle 0-31: In the first 32 cycles, the plaintext nibbles are loaded on to the state register. After this, the round counter resets to 0, and the following operations are repeated 41 times.

Cycle 0-15: Before this set of cycles start, the left branch of the state, resides in the storage elements marked 37 to 20, and the right branch in those marked 17 to 00, as shown in Figure 5.6. In 17 to 00, we need to rotate the right branch by 6 nibbles. This is done as follows: a circular shift is performed for 16 cycles, which is somehow arrested for 10 cycles, to achieve the equivalent functionality of 6 nibble rotation. The 16 nibble flip-flops are divided into 3 groups of 10, 1, 5 (00 to 11, 12 and 13 to 17). An internal circular rotation of nibbles takes

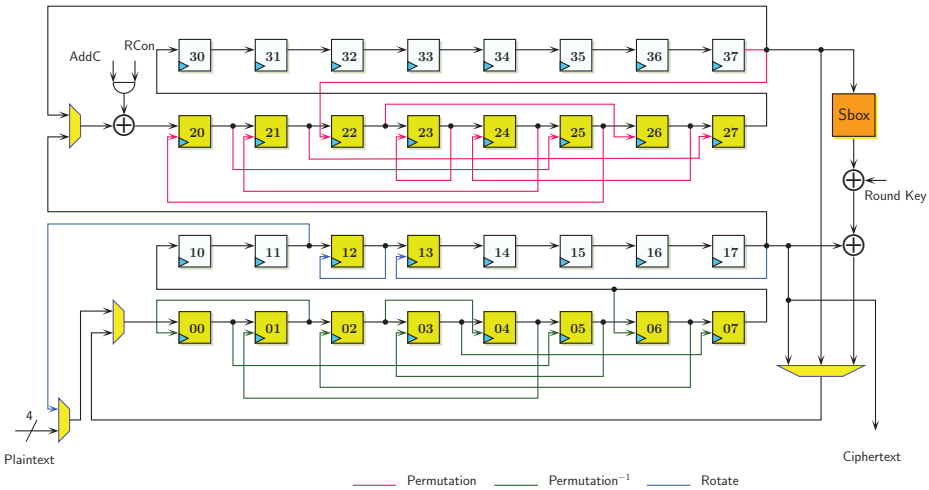


Figure 5.6: Nibble serial architecture for WARP. The filter that feeds the permuted roundkey is omitted in the diagram.

place for 10 cycles within each group. Since 10, 1 and 5 are divisors of 10, this rotation effectively executes the identity transformation on the right branch. Thereafter, a normal circular rotation over the entire set of 16 nibbles (00 to 17) occurs for the next 6 cycles, thus achieving the required functionality.

In the upper half, the shuffling denoted by the permutation π is performed on the left branch (note that the order of shuffle and addkey/sbox is interchangeable). We further take advantage of the fact that π can be defined in terms of the 8-element permutation function $\pi' = \{3, 7, 6, 4, 1, 0, 2, 5\}$ over $[0, 7]$ and $[8, 15]$ (i.e. $\pi[i] = \pi'[i]$ if $i < 8$ and $8 + \pi'[i - 8]$ otherwise). This being so, only the nibbles marked $20 - 27$ need to be scan flip-flops. We perform a circular motion over the left branch nibbles (20 to 37) for these 16 cycles (AddC is set to 0 for this purpose), with the select signal controlling the scan flip-flops being SET at cycles 7 and 15. At cycle 7, the most significant nibbles of the left branch reside at the flip-flops marked 26 to 20 and 37. When the scan flip-flops are SET during this cycle, the wiring ensures that at cycle 8, these nibbles are shuffled by π' and stored in 27 to 20. A similar logic applies to the shuffle in cycle 15. At this cycle, the least significant nibbles of the left branch reside at the flip-flops marked 26 to 20 and 37. The SET signal of the scan flip-flops in this cycle ensures shuffling by π' in the next cycle.

Cycle 16-31: The left branch nibbles are driven out of 37 input to the S-box and then xored with the corresponding key nibble. The output is added with the right branch nibbles which are driven out of 17. The nibbles driven out from 37 are driven back into 20 (thereby causing a circular shift of 16 nibbles which is essentially the identity function). The output of the final xor is driven into 00. Thus after cycle 31, the lower flip-flops (17 to 00) thus contain the output of the round function. The upper flip-flops (37 to 20) continue to hold the left branch of the current round (however the nibbles are shuffled with the permutation π executed in cycles 0 to 15).

Cycle 32-47: We need to undo the shuffling of the left branch and then swap the 2 branches. This is done serially over 16 cycles, by a circular rotation over the 32 flip-flop nibbles (37 to 00). The nibbles driven out of 17 are driven into 20, and thus after this set of 16 cycles, the flip-flops in the upper half (37 to 20) will contain the round function output. The nibbles out of 37 are driven into 00 and it is here that the π^{-1} is performed to undo the shuffle. Note that

Table 5.8: Comparison of performance metrics for serial implementations synthesized with STM 90nm Standard cell library. Figures separated by / indicate corresponding metrics for encryption/decryption. *Synthesized with the IBM 130 nm process/Power at 100 KHz

	Degree of Serialization	Area (GE)	Delay (ns)	Cycles	TP_{MAX} (MBit/s)	Power (μW) (@10MHz)	Energy (nJ)
GIFT-128-128	4/32	1455	2.25	714	76.0	61.7	4.40
GIFT-128-128	1	1213	2.46	6528	7.6	40.3	26.30
SKINNY-128-128	8	1638	1.95	840	74.5	79.1	6.64
SKINNY-128-128	1	1110	0.81	6976	21.6	53.8	37.53
SIMON 128/128	1	1077	1.17	4480	23.3	60.5	27.10
Midori 128 (E)	8	1308	4.94	415	62.4	54.4	2.26
Midori 128 (ED)	8	1401	6.08	415/463	50.7/45.5	54.6	2.27/2.53
AES 128 (ED)	8	2060	5.79	246/326	85.7/64.7	129.7	3.19/4.23
AES 128 (E) [102] *	1	1560	-	1776	-	0.823	14.61
AES 128 (ED) [102]*	1	1738	-	1776/2512	-	0.852	14.61/15.13
WARP (E)	4	**871**	2.97	2032	20.2	33.2	6.76
WARP (E)	1	**763**	2.01	8128	7.5	28.4	23.04
WARP (ED)	4	**925**	2.58	2032	23.3	34.6	7.03
WARP (ED)	1	**806**	2.13	8128	7.1	29.0	23.59

in the bottommost row, the scan flip-flops are wired to perform π^{-1}. The select signals are SET in cycles 40 and 47 to perform π'^{-1} over the lower and upper set of 8 nibbles exactly as in cycles 16-31. This not only moves the left branch nibbles to the lower flip-flops but also undoes the shuffle performed in cycles 0-15, and so we are ready to perform the next round computations. Note that the round constants are added to the register 17 in cycles 32 and 33. This completes the round function. Note that since the left-right swap is omitted in the last round, the ciphertext is output from the flip-flop marked 17 rather than 37.

More circuit details of bit-serial and unified architecture for encryption and decryption are presented in Sect. 5.5.4.

5.5.2 Performance Results

In Table 5.8, we compare the hardware performances of the serial implementations of WARP with other lightweight ciphers, with 128-bit block size and providing 128-bit security. Unless otherwise specified, for all the designs in the table, the following design flow was adhered to. The ciphers were first implemented in VHDL and a functional simulation was done using the *Mentorgraphics Modelsim* software. Thereafter the design was synthesized using the Standard cell library of the STM 90nm CMOS logic process (CORE90GPHVT v 2.1.a) with the Synopsys Design Compiler, with the compiler flag set to `compile_ultra`. A timing simulation was done on the synthesized netlist with 1000 test vectors. The switching activity of each gate of the circuit was collected while running post-synthesis simulation. The average power was obtained using Synopsys Power Compiler, using the back annotated switching activity.

Serial implementations are deployed when area is one of the primary metrics to be optimized. As can be seen from Table 5.8, WARP performs well as far as area is concerned, when compared with other ciphers with similar security level. As in [11, 29], we used multiplexers to filter round keys, instead of a register, which saves us 100 to 150 GE of silicon area. The encrypt-only (E) bit-serial version of WARP occupies only 763 GE which is the lowest reported at this security level. Note that for a fair comparison, all the designs in Tables 5.8, 5.9 were implemented from scratch except the ones marked by an asterisk.

5.5.3 Round Based and Round Unrolled Designs

While serial implementations are useful to construct low area architectures, round based and round unrolled architectures offer a lot of benefits such as good energy performances, in addition with

Table 5.9: Comparison of performance metrics for round based implementations synthesized with STM 90nm Standard cell library (1R, 2R, 4R refer to 1, 2, and 4 round unrolled circuits).

	Area (GE)	Delay (ns)	Cycles	TP_{MAX} (GBit/s)	TP_{MAX}/Area MBit/(s·GE)	Power (μW) (@10MHz)	Energy (pJ)
GIFT-128-128	1997	1.85	41	1.611	0.826	116.6	478.1
SKINNY-128-128	2104	1.85	41	1.611	0.784	132.5	543.3
SIMON 128/128	2064	1.87	69	0.937	0.465	105.6	728.6
Midori 128(E)	2522	2.25	21	2.649	1.076	89.2	187.3
Midori 128(ED)	3661	2.44	21	2.443	0.683	108.7	228.3
AES 128	7215	3.83	11	3.113	0.442	730.3	803.3
WARP (1R) (E)	1187	2.05	42	1.418	1.223	55.5	233.2
WARP (1R) (ED)	1390	1.74	42	1.671	1.231	59.5	250.0
WARP (2R) (E)	1456	1.95	22	2.911	2.047	58.4	**128.5**
WARP (2R) (ED)	1824	2.67	22	2.126	1.193	69.9	**153.7**
WARP (4R) (E)	2223	3.25	12	3.334	1.536	117.5	141.0
WARP (4R) (ED)	3075	3.93	12	2.758	0.918	177.4	212.9

reasonably good area and throughput performances. In [14], the authors studied a number of block ciphers and came to the conclusion that round based or 2-round unrolled implementations tend to be the most energy efficient configurations for block ciphers.

For WARP, the round based configuration would need to filter the upper or the lower key half in successive rounds. Thus a multiplexer is necessary for this filtering. In contrast a 2-round unrolled configuration performs 2 round function computations in a single clock cycle. Such a configuration would have circuits for 2 round functions placed serially one after the other. This obviates the use of a multiplexer to filter any round keys, as it is clear that the first round function block can simply use the upper key half and the second block can similarly use the lower half. Thus a 2-round unrolled circuit would consume proportionately lesser resources than a round based circuit both in terms of area and energy. Similar arguments can be made about odd and even round unrolled circuits for WARP. We experimented with 3 configurations for WARP: the round based, the 2 round and the 4 round unrolled circuits. The simulation results along with a comparison with other lightweight block ciphers is presented in Table 5.9. Indeed, in terms of energy, the 2-round unrolled configuration is the best and is around 30% better with respect to the one round configuration of Midori 128, a block cipher, which is the most energy efficient block cipher reported in literature. Note that WARP has odd number of rounds: this means that any even round unrolled implementation will do some redundant computation in the final cycle. For example, a 2-round unrolled implementation will need to operate 21 cycles, to execute 41 round functions: the final cycle performs one additional round function. This amounts to wastage of energy in the final cycle: however this is a small fraction of the total energy consumed (for WARP it is less than 1% of the total energy consumed). Figure 5.7 further shows a breakdown of area occupied by the corresponding components of the circuit. Sect. 5.5.4 also describes a 1st order threshold implementation of the WARP circuit.

5.5.4 More Details about Hardware Implementations

5.5.4.1 Bit Serial Architecture The nibble serial architecture can be converted to a bit serial architecture, with some simple circuit-level transformations. The first is explained in Figure 5.8.

Any nibblewise scan flip-flop can be serialized as shown in Figure 5.8, so that only one scan flip-flop per nibble is utilized. Whereas in the nibble serial architecture, the circuit can transfer one of the 2 nibble signals in one clock cycle, the same can be done over 4 cycles in the bit-serial architecture. Thus the bit-serial circuit can perform the same set of round operations in $48 \times 4 = 192$ cycles, in 3 sets of 64 cycle operations as in the nibble serial circuit. We save $18 \times 3 = 54$ scan flip-flops in the bit-serial architecture, and also 4-bit xor gates and multiplexers

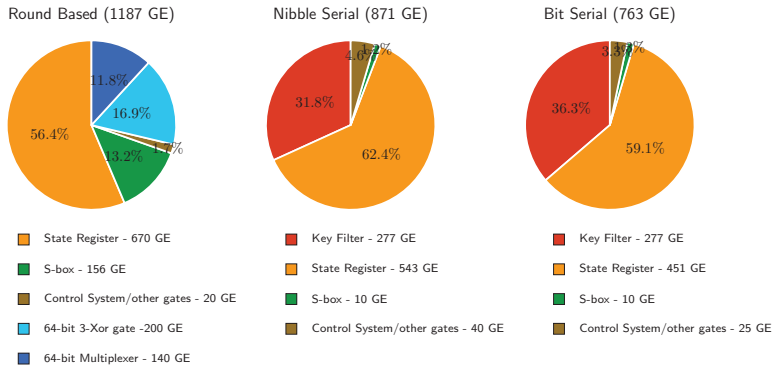


Figure 5.7: Breakdown of component-wise area figures for 3 versions of WARP. Nibble and Bit-serial circuits require lesser scan flip-flops which require more area

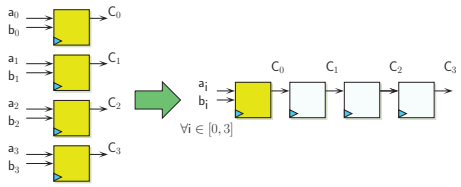


Figure 5.8: Nibble to bit serial transformations can be replaced with corresponding single bit gates.

5.5.4.2 Unified Circuit for Encryption and Decryption Implementing the functionalities of encryption and decryption (ED) on the same circuit can be beneficial in some instances. Various modes of operations like CBC, XTS, OCB and COLM [6], that use block ciphers as the underlying primitive, require access to both its encryption and decryption functionalities. Thus it is useful to have an implementation that achieves both functionalities of a block cipher with minimal overhead. There are several features in the Feistel network structure, that make it easier to construct the ED architecture. Some of them are as follows:

1. SPN structures generally require involutive S-boxes to ensure efficient ED implementation [11,29]. If not, they require the circuits for the forward and inverse S-box to be implemented together, which increases area [16]. However the inverse round function in Feistel networks can be described with the forward S-box only. This gives us more freedom to search for S-boxes with lightweight characteristics.

2. Some SPN block ciphers, e.g., [29] require the decryption key to be equal to $L \cdot K$, where L denotes the matrix that forms the linear layer of the cipher. Thus additional circuit for matrix multiplication is required. Also a multiplexer is required to filter these keys for encryption/decryption. In the architecture for WARP, this is not necessary.

3. Let F_K denote the function that performs S-box function and the roundkey addition on the left branch. Then by slight abuse of notation we can write the round function as

$$Y_a = P(F_K(X_a)) \oplus (X_b \lll 6), Y_b = X_a \tag{5.1}$$

where P is the function that performs the nibblewise shuffle of the left branch by moving the $i-th$ nibble to $\pi[i]$. Then it is easy to see that the inverse round function is $X_a = Y_b, X_b = (P(F_K(Y_b)) \oplus (Y_a)) \ggg 6$. However we do omit the left-right swap in the last round, and

as a result, decryption can be computed by iterating the following round function 40 times, followed by a "swapless" final round:

$$X_a = (P(F_K(Y_a)) \oplus (Y_b)) \ggg 6, X_b = Y_a. \tag{5.2}$$

Equations (5.1) and (5.2) are similar except that in encryption, the right branch is left rotated by 6 nibbles before addition, whereas in decryption, rotation done is after xoring left and right branches, this time by 6 nibbles towards the right. Since WARP uses each half of the master key in alternate rounds for key addition, it has been designed to have odd number of rounds. This means the first and last round encryption keys are the same, which implies that the encryption and decryption uses the left and right halves of the key in the same order.

Thus the only real overhead in the ED circuit for WARP is to accommodate left and right rotation by six nibbles in different times of the decryption cycle, and arrange for round constants to be generated in the reverse direction, which only requires some strategically placed multiplexers to accommodate the timing of these operations in the decryption cycle.

In essence, one approach would be to not rotate the right branch during cycles 0 to 15, and do rotation only during cycles 31 to 47 when the xoring of right and left branches has been completed. However, this approach is slightly problematic to adopt, as cycles 31 to 47 are used to not only swap left and right branches but also to apply π^{-1} to the left branch as it is being moved to bottom rows of flip-flops. In such a situation the bottom rows cannot accommodate two different types of permutation operations at the same time.

As a result, we need to exercise some fine-grained control over the ED circuit. For decryption, we rotate the right branch by 10 nibbles left in cycles 0-15 (same as 6 nibbles right rotation), although this rotation is not required as per Equation (5.2). Thus to maintain functionality, in cycles 16-31, we drive nibbles out through register 11 to do xor between left and right branches (this was done through flip-flop 17 during encryption). After xor, the incoming nibbles are driven in through flip-flop 12 (this was done through 00 during encryption). This method has the added advantage that after round 31, the flip-flops $17 - 00$ already contain $(P(F_K(Y_a)) \oplus (Y_b)) \ggg 6$. This allows us to have the decryption operations in cycles 32-47 exactly the same in encryption.

For completeness, we discuss two more issues. First, for decryption we choose, cycles 0 and 1 for round constant addition, as this operation has to precede the non-linear operations. To rotate left by 10 nibbles, we need to freeze rotation for 6 cycles. Like in encryption, we divide the bottom row into groups of 6, 6, 2, 2 flip-flops and do internal rotation in these for 6 cycles. To accommodate this operation we need to replace 3 normal flip-flop nibbles with scan flip-flop nibbles.

5.5.4.3 Threshold Implementations The s-box belongs to the cubic class $\mathcal{C}_{266}$ as per the classification in [46] and as such it can be decomposed into 2 quadratic s-boxes $F \circ G$, where

$$G = [\texttt{0, f, 6, 1, 3, 8, d, e, 4, b, 2, 5, 7, c, 9, a}],$$
$$F = [\texttt{c, 3, 1, e, 8, 5, d, 0, b, 4, 6, 9, 2, f, 7, a}]$$

Since a minimum of $d+1$ shares are required to implement the 1st order threshold implementation (TI) of a degree d s-box, we can thus implement a 3 share 1st-order TI in the manner shown in [143]. The idea is to implement the TI of G and F separated by a register bank in between, which suppresses the glitches produced by the TI of G.

Since the s-box has degree 3, a straightforward 4-share TI can also be implemented using a direct sharing approach. With regards to circuit architecture, the 4 share versions would only consist of 4 copies of the unshared circuit combined through a shared s-box. The 3-share circuit is slightly complicated owing to the fact that the shared G, F functions have to be executed one

Table 5.10: Comparison of performance metrics for serial implementations synthesized with STM 90nm Standard cell library. (RB denotes round based circuit, 3s, 4s denotes circuits with 3, 4 shares respectively) *Synthesized with the UMC 180nm process/Power at 100 KHz. **Synthesized with the IBM 130nm process/Power at 100 KHz

	Degree of Serialization	Area (GE)	Delay (ns)	Cycles	TP_{MAX} (MBit/s)	Power (μW) (@10MHz)	Energy (nJ)
PRESENT-80 (3s)* [143]	4	2282	-	547	-	5.1	28.16
SKINNY-128-128** (3s) [28]	8	3780	1.63	872	90.0	-	-
WARP (3s)	4	**2288**	3.11	2032	19.3	99.9	20.29
WARP (3s)	1	**1964**	2.54	8128	5.9	87.0	70.72
WARP (3s)	RB	**6033**	2.73	83	545.3	232.5	1.93
WARP (4s)	4	**3363**	3.12	2032	19.3	145.7	29.61
WARP (4s)	1	**3060**	3.26	8128	4.6	136.6	111.03
WARP (4s)	RB	**15761**	3.38	42	880.9	703.3	2.95

after the other. We implemented it in the manner shown in Figure 5.9. In the unprotected circuit described earlier, the S-box layer is computed in cycles 16 to 31. And so in the shared circuit, we implement the shared function G in cycles 15 to 30 and the shared function F in cycles 16 to 31. However this creates another problem, as the entire left branch is overwritten by the output of the shared G layer before being fed back into register 20. Since the current left branch is required to serve the role of the right branch in the subsequent round, we need to invert the G layer before we proceed to the next round. This is done by implementing a shared implementation of the quadratic s-box G^{-1} between registers 20 and 21, which is operated from cycles 17 to 32.

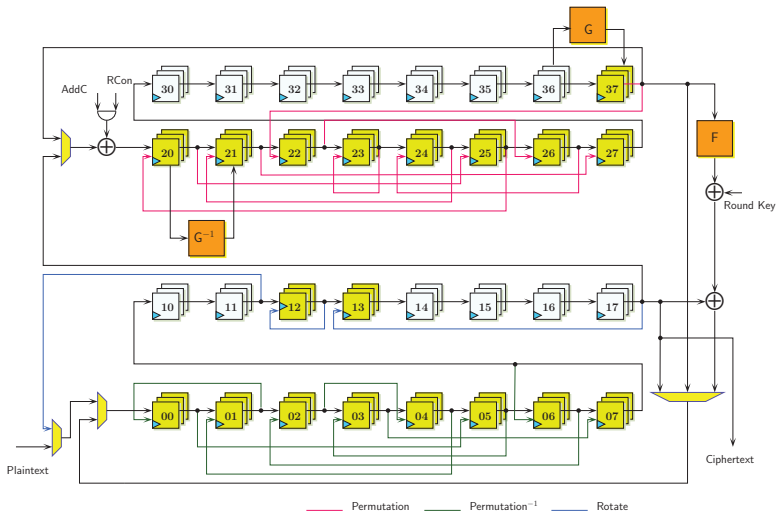


Figure 5.9: Sketch of the 3 share nibble serial architecture for WARP

Table 5.10 shows the performance results for the 3-share and 4-share implementations of WARP. The smallest 3-share implementation stands at 1964 GE which is smaller than known implementations of SKINNY and PRESENT (although these are computed at different level of serialization).

5.6 Software Performance

5.6.1 On 8-bit AVR Microcontrollers

The design of WARP makes it flexible to make trade-offs to achieve various performance characteristics on 8-bit AVR. Applying different implementation choices results in different trade-offs between ROM, RAM, and execution time. Sect. 5.7.1.1 presents the details of our implementations. Table 5.11 and 5.12 summarizes the results and comparison with available results of existing designs with same parameters. It can be seen, on one end of the spectrum, WARP consumes minimized RAM and competitively low ROM; On the other end, it achieves relatively good performance regarding CPU cycles without consuming too much ROM. Compared with other ciphers, there are two characters on the architecture of WARP, which are the almost self-reciprocal structure and the extreme simplicity of the key schedule. These two characters make the code size competitively small when all encryption, decryption, and key-schedule are required to be implemented. The lower level choices on the nibble-oriented operations make it flexible to make trade-offs.

5.7 On High-end Processors

The nibble-orientate character of WARP enables implementations of it fit neatly with a Single Instruction Multiple Data (SIMD) instruction commonly seen on modern CPUs. This SIMD instruction performs a vector permutation providing a look up table representation of the permutation offsets, which are called Vector Permutation Instruction (VPI) [168]. Examples of CPUs equipped with VPIs includes Motorola AltiVec (the `vperm`), Intel and AMD x86-64 (the `(v)pshufb`), and ARM NEON (the `vtbl`). Concretely, take `(v)pshufb` that is avaiable on Intel and AMD x86-64 CPUs and was used in our implementations, as an example, denote the source operands by a and b hold in two 128-bit registers, and let the destination operand be c. The resulted value $c[i]$ in the i-th byte position of c will equal $a[b[i]]$ for $0 \leq i < 16$ if the most significant bit of $b[i]$ is not equal to 1, and 0 otherwise. The byte $b[i]$ in b encodes the index of the byte in a that will be moved into $c[i]$. Thus, only the lower 4 bits of $b[i]$ are used to encoding the index. For Intel and AMD x86-64 CPUs, the concrete VPI is named `(v)pshufb` (which were used in our implementations). Both the parallel 4-bit S-box and the nibble shuffle operation can be implemented using `(v)pshufb`. The difference lies in that for the former, the inputs to the S-boxes act as the index, whereas for the latter, the table representation of the permutation act as the index. Thus, the round function of WARP can be fully implemented using a few `(v)pxor` and `(v)pshufb`. In Sect. 5.7.1.2, we present the details of our implementations of WARP using SIMD instructions on x64 CPUs. Our benchmark results of WARP, together with that of two ciphers that are also designed targeted at hardware, i.e., SIMON and SKINNY, are reported in Figure 5.10.

The software performance of WARP on high-end processors has the following advantages. First, apart from mode of operations that can be parallelized, for those that cannot, WARP also provides competitive performance, because the single-block implementation of WARP can be very fast. Besides, for those modes that can be parallelized, the latency of WARP can be very small, because the required number of message blocks to achieve the optimal performance is relatively small. Second, in the scenario where a server communicates with many sensors using different keys, WARP can be very fast, because there is no heavy key schedule.

The source codes for our software implementations can be found via `https://github.com/WARP-Block-Cipher/Software`.

Table 5.11: Different performance characteristics of WARP on 8-bit AVR

Unroll	Features	Function	ROM [B]	RAM [B]	Time [cyc.]	Speed [cpB]
One-Round	One-Sbox ROM	ENC	(908 - 46) 862	(160 - 160) 0	56408	440.69
		DEC	(914 - 46) 868	(160 - 160) 0	56762	443.45
		ENC+DEC	(996 - 46) 950	(160 - 160) 0	113170	884.14
	One-Sbox RAM	ENC	(956 - 46) 910	(176 - 160) 16	50554	394.95
		DEC	(962 - 46) 916	(176 - 160) 16	50908	397.72
		ENC+DEC	(1044 - 46) 998	(176 - 160) 16	101462	792.67
	Two-Sbox ROM	ENC	(1084 - 46) 1038	(160 - 160) 0	40664	317.69
		DEC	(1090 - 46) 1044	(160 - 160) 0	41018	320.45
		ENC+DEC	(1172 - 46) 1126	(160 - 160) 0	81682	638.14
Two-Round	Two-Sbox ROM	ENC	(1404 - 46) 1358	(160 - 160) 0	36504	285.19
		DEC	(1406 - 46) 1360	(160 - 160) 0	36506	285.20
		ENC+DEC	(1516 - 46) 1470	(160 - 160) 0	73010	570.39
	Two-Sbox RAM	ENC	(1264 - 46) 1218	(416 - 160) 256	34348	268.34
		DEC	(1266 - 46) 1220	(416 - 160) 256	34350	268.36
		ENC+DEC	(1376 - 46) 1330	(416 - 160) 256	68698	536.70

5.7.1 More Details of Software Implementations

5.7.1.1 Details of Software Implementations on 8-bit AVR In this section, we report various performance of our software implementations of WARP on Atmel 8-bit AVR. The implementations were written in assembly and compiled using AVR macro assembler 2.2.7 in Atmel Studio 7.0. The considered performance metrics include the code sizes (ROM), RAM usage, and execution time (CPU cycles), which were also measured using Atmel Studio 7.0. All implementations followed the equivalent LBlock-type structure (see Figure 5.4 and 5.11b). For detailed implementation, we considered the following choices:

1. One-round or two-round unrolling: combining two rounds can save the shuffle operation between left and right branches, trading ROM for CPU cycles. Note that the nibble-shuffle π on the left branch requires no additional procedure when hard-coding the correspondence between the XORed nibbles.

2. One-S-box or two-S-box combining: combining two 4-bit S-boxes to an 8-bit can reduce times of memory accesses, trading ROM or RAM for CPU cycles.

3. Storing the look up table (LUT) of the S-box in RAM or ROM: storing the LUT in ROM can minimize the consumption of RAM; the downside is on the one extra CPU cycle per memory access.

Applying different choices results in different trade-offs between ROM, RAM, and execution time. Table 5.11 summarizes the evaluation results of our various implementations; Table 5.12 presents two representative results together with the available results of other block-ciphers (with 128-bit block and key) for comparison. It can be seen that, on one end of the spectrum, the implementation of WARP can consume minimized RAM (excluding the scenario-specific RAM data, such as data to be encrypted, master keys, and initialization vectors.) and competitively low ROM. On the other end of the spectrum, the implementation can achieve relatively good performance regarding CPU cycles without consuming too much ROM. Compared with other ciphers, there are two characters on the architecture of WARP, which are the almost self-reciprocal structure and the extreme simplicity of the key schedule. These two characters make the code size competitively small when all encryption, decryption, and key-schedule are required to be implemented. The lower level choices on the nibble-oriented operations make it flexible to make trade-offs.

5.7.1.2 Details of Software Implementations on x64 CPUs We implemented the serial processing of message block using 128-bit registers, the parallel processing of double or more blocks using 256-bit registers. Because of the byte-orientate character of the `(v)pshufb`

Table 5.12: Performance of block ciphers (128-bit block and 128-bit key) on 8-bit AVR

Cipher	Block [b]	Key [b]	ROM [B]	RAM [B]	Time [cyc.]
AES	128	128	3000	(406 - 160) 246	58973
LEA	128	128	1650	(629 - 160) 469	61755
SKINNY	128	128	1124	(545 - 160) 385	77451
SPARX	128	128	1726	(751 - 160) 591	84390
WARP [1R]	128	128	1126	(160 - 160) 0	81682
WARP [2R]	128	128	1330	(416 - 160) 256	68698

The target device is ATmega128; The scenario is encryption/decryption of 128 bytes of data in CBC mode [75]. For ROM, that consumed by the main function for initializing data and calling the enc/dec functions are subtracted. For RAM, that required for storing the data to be processed, the master key, and the initialization vector are subtracted. WARP [1R] is for the one-round-based implementation storing the LUT of two-S-box in ROM. WARP [2R] is for the two-round-unrolled implementation storing the LUT of two-S-box in RAM. Results of other ciphers are from https://www.cryptolux.org/index.php/FELICS_Block_Ciphers_Brief_Results.

instruction, in each byte of the 128- or 256-bit registers, only the lower 4 bits are used. Half of the 32 branches of one (resp. two) block are stored within one 128-bit (resp. 256-bit) register.

The detailed implementation choices we considered are as follows.

1. Using equivalent form: similar to TWINE [168], it is possible to transform WARP into an equivalent form (see Figure 5.12), in which only half branches go through a nibble shuffle in each round[10]. We denote a shuffle of 16 nibbles by "half shuffle". Four different half shuffles are required for round functions because the shuffle parameters form a loop every four encryption rounds. Additionally, eight different shuffles are required to get equivalent round keys because the shuffle parameters for round keys form a loop every eight rounds. Details of the shuffle parameters can be seen in Figure 5.12.

2. Using three-operand instructions in AVX2: The two-operand instruction, pshufb in SSSE3, performs in-place shuffles and thus, original data in the first operand will be destroyed. Whereas, the three-operand instruction vpshufb, keeps the source data, and stores the result in a third register. To implement parallel S-boxes using pshufb/vpshufb, the table representation of the S-box must be in the first source operand. Using pshufb, this table representation will be replaced by the result, thus inevitably requires reloading; Whereas, using vpshufb, it will not be destroyed and thus avoid the frequent reloading.

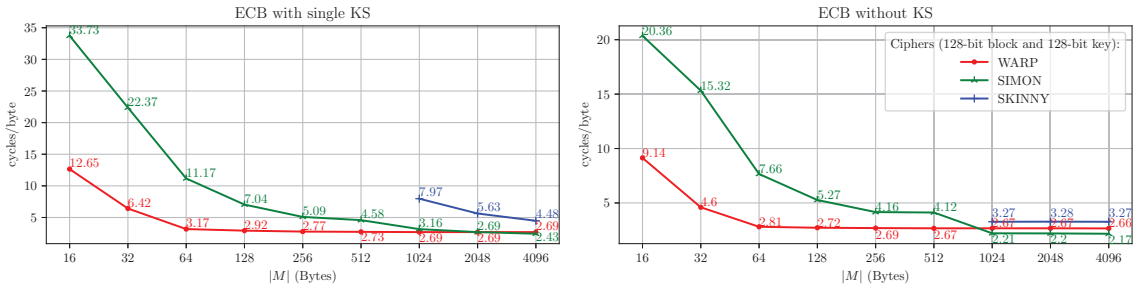
3. Taking advantage of the pipelined execution unit on modern CPUs: in our implementation, we provide additional options on parallelism (besides the double-block parallelism) – compute an atomic instruction (e.g., vpxor for round key addition) on quadruple/octuple data blocks, and only then continue to computing the subsequent atomic instruction on for quadruple/octuple data blocks. This is because, on modern CPUs, the hardware that support the SIMD instructions is pipelined, which allows independent SIMD instructions to be dispatched before the completion of one instruction. By experiment, processing quadruple/octuple blocks using two/four AVX2 instructions on 256-bit registers achieves more performance benefit.

Different processors and different methods for the measurement may cause non-negligible influence on the results, which eventually makes it difficult to do a fair comparison. Considering this, we chose two ciphers, i.e., SKINNY and SIMON, to do benchmarks on the same processor using the same method. Note that these two ciphers are also designed targeted at hardware and at the same time, have competitive software performance; besides, their optimized source codes are publicly available. Our benchmark results are reported in Figure 5.10. For WARP, we also provide more detailed performance evolution with the length of the messages in Table 5.13.

[10]Although a swap is also required between half branches and another half at the end of each round, it can be skipped by variable renaming and unrolling even rounds.

Table 5.13: Software performance profile of WARP with various message length (including the time took by packing/unpacking messages)

$\|M\|$ (bytes)	Haswell (Parallelization)				Skylake (Parallelization)			
	1	2	4	8	1	2	4	8
16	11.20	11.18	11.24	11.21	8.41	8.87	9.00	9.14
32	8.71	5.68	5.65	5.67	8.15	4.60	4.52	4.60
64	8.61	4.48	3.01	2.93	8.15	4.25	2.78	2.81
128	8.55	4.40	2.81	2.81	8.18	4.16	2.71	2.72
256	8.51	4.38	2.79	2.72	8.20	4.17	2.72	2.69
512	8.50	4.36	2.77	2.70	8.18	4.12	2.70	2.67
1024	8.50	4.36	2.76	2.67	8.18	4.09	2.71	2.67
2048	8.50	4.36	2.77	2.67	8.17	4.08	2.71	2.67
4096	8.56	4.43	2.76	2.67	8.17	4.08	2.71	2.66



Source code for SKINNY and SIMON (versions with 128-bit block and 128-bit key) were adapted from [111, 174]. Because that of SKINNY only support 64-block parallel processing, results for short message are not available. We used GNU g++ 5.5.0 with `-O3 -mavx2` options to compile. The processor is Intel(R) Core(TM) i7-6700 (Skylake). We turned off hyper-threading and disabled Turbo Boost. The timing method used was that in `http://github.com/BrianGladman/AES`. The instruction is `rdtsc`. We used `time_enc16()` evaluating the average time using 10000 samples of messages of a particular length.

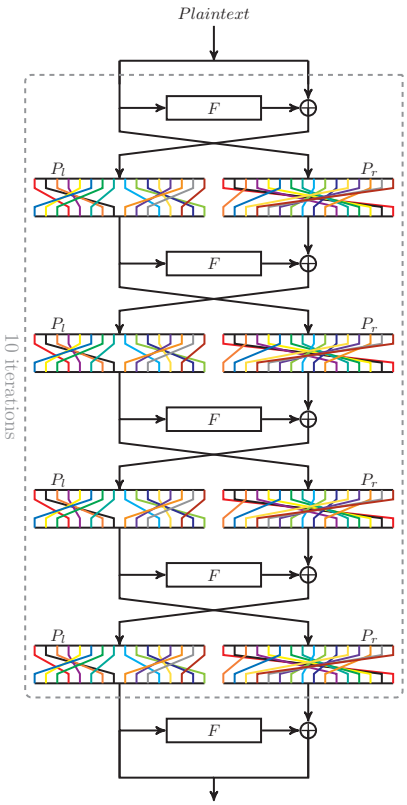
Figure 5.10: Software performance of WARP, SIMON and SKINNY on the same processor.

5.7.1.3 The Equivalent Forms Used in Software Implementations

Figure 5.11b shows an equivalent form of WARP in `LBlock-like` structure used in our implementations for 8-bit AVR. Figure 5.12 shows the equivalent form of WARP used in our SIMD implementations.

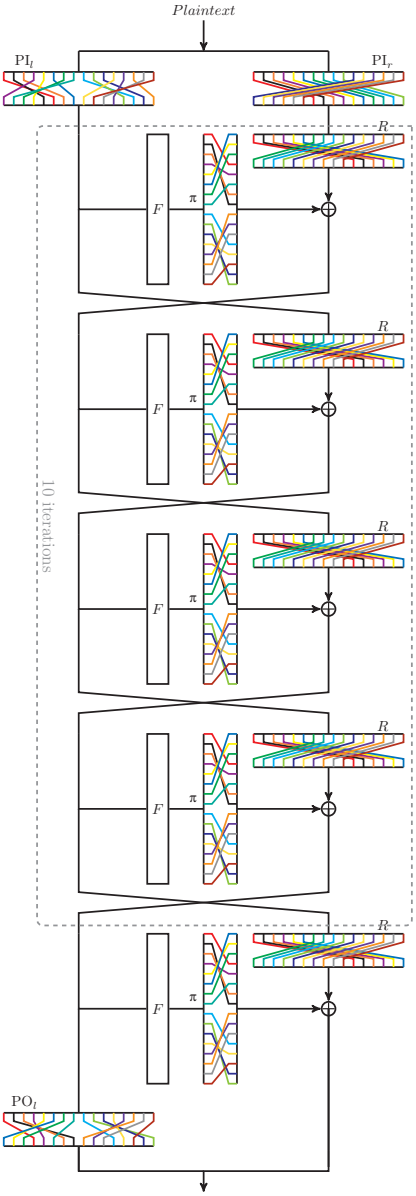
5.8 Conclusion

We have presented a 128-bit lightweight block cipher WARP. The design of WARP is based on a variant of Type-2 GFN, combined with an improved shuffle over 32 nibbles to boost the diffusion. The primary goal is to achieve a small-footprint 128-bit block cipher, both for encryption-only and unified ED circuits. This has been achieved by carefully choosing the components of GFN. We provided a comprehensive hardware implementation results. They show that WARP is the smallest 128-bit block cipher in the most of typical implementation strategies. Moreover, WARP is very competitive in energy-efficient implementation. Besides, the software of WARP on 8-bit microcontrollers can achieve competitively small code size and extremely low RAM consumption, with acceptable execution time. Finally, WARP is very efficient on software implementation using SIMD on high-end processors. Indeed, our experimental results suggest that, for relatively short inputs, WARP is faster than other hardware-oriented lightweight ciphers, which is a desirable feature when the block cipher is operated in a serial mode.



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

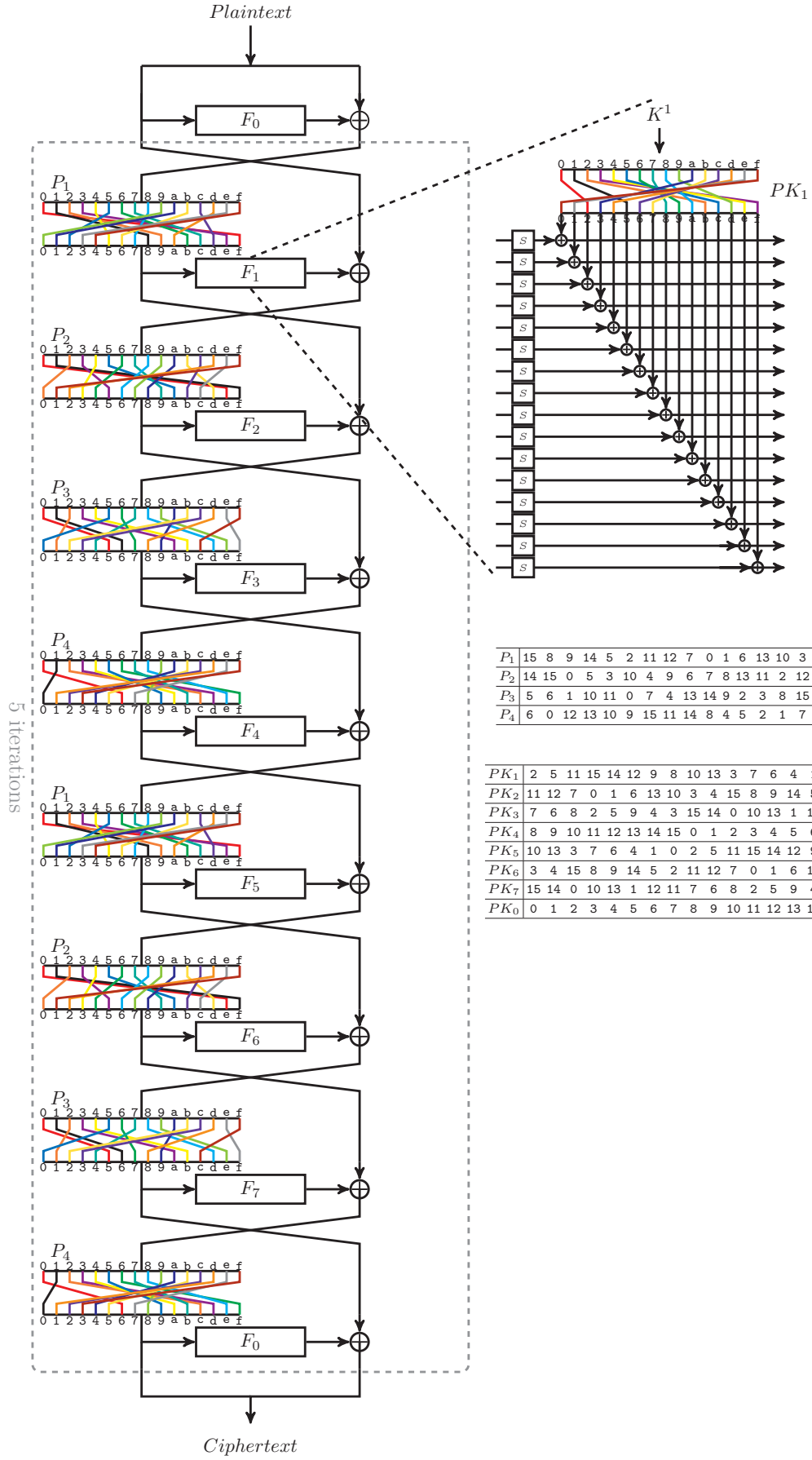
(a) An equivalent form by separating even and odd nibbles to left and right half branches



π	3	7	6	4	1	0	2	5	11	15	14	12	9	8	10	13
R	10	11	12	13	14	15	0	1	2	3	4	5	6	7	8	9
PI_l	5	4	6	0	3	7	2	1	13	12	14	8	11	15	10	9
PI_r	6	7	8	9	10	11	12	13	14	15	0	1	2	3	4	5
PO_l	3	7	6	4	1	0	2	5	11	15	14	12	9	8	10	13

(b) An equivalent form in `LBlock-like` structure

Figure 5.11: Equivalent forms of WARP



P_1	15	8	9	14	5	2	11	12	7	0	1	6	13	10	3	4
P_2	14	15	0	5	3	10	4	9	6	7	8	13	11	2	12	1
P_3	5	6	1	10	11	0	7	4	13	14	9	2	3	8	15	12
P_4	6	0	12	13	10	9	15	11	14	8	4	5	2	1	7	3

PK_1	2	5	11	15	14	12	9	8	10	13	3	7	6	4	1	0
PK_2	11	12	7	0	1	6	13	10	3	4	15	8	9	14	5	2
PK_3	7	6	8	2	5	9	4	3	15	14	0	10	13	1	12	11
PK_4	8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
PK_5	10	13	3	7	6	4	1	0	2	5	11	15	14	12	9	8
PK_6	3	4	15	8	9	14	5	2	11	12	7	0	1	6	13	10
PK_7	15	14	0	10	13	1	12	11	7	6	8	2	5	9	4	3
PK_0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Figure 5.12: Another equivalent form using four permutations for round functions and eight permutations for round keys (omitted addition round constants)

6 Orthros: A Low-Latency PRF

We present Orthros, a 128-bit block pseudorandom function. It is designed with primary focus on latency of fully unrolled circuits. For this purpose, we adopt a parallel structure comprising two keyed permutations. The round function of each permutation is similar to Midori, a low-energy block cipher, however we thoroughly revise it to reduce latency, and introduce different rounds to significantly improve cryptographic strength in a small number of rounds. We provide a comprehensive, dedicated security analysis. For hardware implementation, Orthros achieves the lowest latency among the state-of-the-art low-latency primitives. For example, using the STM 90nm library, Orthros achieves a minimum latency of around 2.4 ns, while other constructions like PRINCE, Midori-128 and QARMA_9-128-σ_0 achieve 2.56 ns, 4.10 ns, 4.38 ns respectively.

6.1 Introduction

6.1.1 Low-Latency Encryption

Lightweight cryptography is a subfield of symmetric-key cryptography to study cryptographic core functions usable under strong resource constraints. Hardware circuit size is a typical metric, and there are numerous proposals such as GIFT [20], KATAN [71], LED [90], Piccolo [161], PRESENT [52], and SIMON [24] particularly perform well on this metric. Some other metrics exist, and among them, latency has been increasingly receiving attention. Latency affects the response time of encryption or authentication, and a small latency is highly desirable for applications that require instant response, such as encryption of memory bus, storage systems, automotive communication and industrial control network. Gaining throughput is generally possible with common signal processing techniques (pipelining and parallel processing), while achieving a low latency remains a challenge [108].

To our knowledge, the first lightweight block cipher with explicit focus on latency is PRINCE proposed by Borghoff et al. [53]. PRINCE is a 64-bit block cipher comprising multiple round functions to significantly reduce the number of rounds while keeping a moderate complexity of each round. Another proposal is QARMA proposed by Avanzi [9], which is a family of low-latency tweakable block ciphers (TBCs) [118]. It adopts the design strategy of PRINCE. Mantis [27] is another family of low-latency TBCs. Midori is a family of block ciphers proposed by Banik et al. [11]. It primary aims to reduce energy, however its latency is also quite small.

The current work on low-latency encryption focused on invertible primitives, i.e., (tweakable) block ciphers. We started with a question whether this is an exclusive approach – namely, whether we can do better by not requiring an invertible primitive. Motivated by this question, we initiated a study on designing low-latency (non-invertible) pseudorandom function (PRF) consisting of parallel keyed permutations. We study a sum of two block ciphers denoted as $C = E_K(M) \oplus E'_K(M)$, where $E, E' : \mathcal{K} \times \mathcal{M} \to \mathcal{M}$ are different n-bit block ciphers with a key space $\mathcal{K}$ and a message space $\mathcal{M} = \{0,1\}^n$. Since E and E' can be computed in parallel, the critical path length of a fully unrolled circuit is a maximum of them instead of the sum. The resulting function has n-bit block and is not invertible in general.

The sum of permutations is indeed not new and it has been adopted in the designs of RIPEMD-160 [76] and Grøstl [83]. In addition, the sum of permutations has also been extensively studied in the context of provable security (see Section 6.3.1). In particular, the result of Dai et al. [70] suggests that it can ideally achieve n-bit PRF security, i.e., indistinguishable from a truly random function with $O(2^n)$ complexity. However this requires that E_K and E'_K behave as computationally-secure block ciphers, more formally, (computationally-)independent *pseudorandom permutations (PRPs)*. Instead of requiring this, we explore the setting that E and E' are rather weak as a stand-alone block cipher, using a small number of very simple rounds. The point is that the outputs of

E and E' are never given in clear, hence we can hope that both can cover each weakness, and consequently the sum of them can tolerate dedicated attacks as a PRF.

6.1.2 Our Design

Based on the aforementioned considerations, we present Orthros, which is a 128-bit block pseudorandom function (PRF) with a 128-bit key. The overall structure of Orthros is a sum of two SPN-type keyed permutations called Branch1 and Branch2. The round functions of Orthros are based on Midori. It is already suitable to low-latency ciphers, however we performed a thorough study on it and showed that we can further improve latency by adopting new permutation layers and S-boxes.

In particular, we propose a hybrid use of bit and nibble permutations i.e., a bit permutation is used for some rounds and a nibble permutation for the rest, while Midori-128 [11] uses a single linear layer including both of a bit permutation and a nibble permutation. Consequently, each branch of Orthros achieves the 2.5-round full diffusion and attains 64 active S-boxes over 10 rounds, while Midori-128 requires 3 rounds for the full diffusion and 13 rounds for 64 active S-boxes. In addition, the whole Orthros has more than 64 active S-boxes over only 5 rounds. Importantly, this change of linear layers does not require any additional hardware cost in an unrolled implementation.

For PRF, we do not need an involutory S-box unlike Midori and QARMA. This allows us to develop a new 4-bit S-box that offers about twice smaller delay than that of Midori-128 [11] (see Table 6.11 of Page 14).

Since we do not rely on provable security of Sum of Permutations, we carried out an extensive security analysis on not only the components of Branch1 and Branch2 but also the whole Orthros.

6.1.2.1 Motivation for using 128-bit PRF. The lack of invertibility limits applications. For example the classical CBC and XTS (a storage encryption mode) [79] require the decryption routine of a block cipher. However, many popular modes, e.g, CTR, CMAC [78] and GCM [139], do not require the decryption routine (this property is also called inverse-freeness). For these modes Orthros can be used as the cryptographic core. For examples of applications that require low latency, ARM's pointer authentication code (PAC) uses QARMA [144]. PAC is a MAC for the pointer value and the memory context and it is derived by truncating the output of QARMA in an ECB-like mode. The length of PAC ranges from 11 to 31 bits or 3 to 23 bits, depending on a processor feature [144]. The PAC is inserted into a reserved space of the original 64-bit pointer containing the address. This mode is inverse-free, even though QARMA is an invertible primitive.

As another example we consider memory protection schemes based on (a MAC variant of) Merkle Tree. They often use a black-box PRF as a MAC function (e.g, [92]), and a concrete memory encryption scheme inside Intel's SGX [88] adopts inverse-free modes, namely variants of GMAC and GCM. A notable benefit of using a dedicated PRF instead of a PRP is that it can provide a stronger, beyond-birthday-bound (BBB) security depending on the mode. As observed by [127], by changing the component of GCM or CTR from a PRP to a PRF, the provable security immediately improves from $n/2$ to n bits. In such a case, there is a strong incentive to use a PRF from the security perspective.

If we compare our proposal with PRINCE, the larger input and output extend possible applications. In fact, the first example of PAC requires input larger than 64 bits while not exceeding 128 bits. This excludes the application of PRINCE [144, Page 6 (Cryptography)] and makes Orthros usable. In the latter example of tree-based memory protection, a GCM-like authenticated encryption mode with PRINCE will only have 32-bit security while Orthros ensures 128-bit security as described above. We note that the output size is also crucial for 128-bit security in this case, since we need 128 bits of pseudorandom sequence to mask 128-bit universal hash function output.

6.1.2.2 Implementations. We implemented Orthros in four different standard cell libraries along with 2 other constructions Midori-128 and QARMA$_9$-128-σ_0 that have a block size of 128 bits and offer at least 128 bits of security. Across all libraries, Orthros performs around 40 % better than the above designs with respect to **a)** the absolute delay between input/output ports and also **b)** the delay when the circuits are restricted to a certain area/power budget. We even found that Orthros performs marginally better than PRINCE (which has a blocksize of 64 bits and offers (127 - d) bits of security give 2^d plaintext /ciphertext pairs) with respect to the total circuit delay across all libraries. All our implementations are publicly available[11].

6.1.2.3 Related Work. Mennink and Neves [127] proposed a generalized EDMD mode [126] and an instantiation of it by a pair of reduced-round block ciphers, E and E'. The resulting scheme is a PRF of nearly n-bit security [126] if E and E' are PRPs. In [127] they proposed an instantiation based on reduced-round AES, called AES-PRF. It has been analyzed by Derbez et al. [74]. Although a conceptual similarity to us, the generalized EDMD is not suitable for low-latency PRF because it is serial. Besides, they did not consider to use dedicated round functions. In a broader context, reduced-round version of a standard block cipher has been used to build a wide variety of cryptographic functions. Most notably for AES, there are examples such as a MAC function [68, 69], a stream cipher [47], a hash function [89], an authenticated encryption scheme [93], and a tweakable block cipher [21] to name a few.

From the provable security perspective, the n-bit security of sum of two PRPs has been proved [70] as described. Chen et al. [59] studied the sum of Even-Mansour ciphers and proved its tight security of $2n/3$-bit.

6.2 Specification

Orthros is a 128-bit pseudorandom function (PRF) with a 128-bit key, the overview of which is illustrated in Fig 6.1. On the whole, Orthros consists of two SPN-based 128-bit keyed permutations called Branch1 and Branch2, each composed of an S-layer, P-layer, the round-key addition and the constant addition. The S-layer is the parallel application of a 4-bit S-box and the P-layer is a linear transform (bit or nibble permutation, followed by a matrix multiplication). Moreover, two key scheduling functions called KSF1 and KSF2 based on two different bit permutations are exploited in Branch1 and Branch2, respectively.

In Orthros, a 128-bit plaintext M[12] is first copied to two 128-bit internal states X^1 and X^2. Then X^1 and X^2 are respectively given to Branch1 and Branch2. The 128-bit ciphertext C is an XOR of the outputs of Branch1 and Branch2. More details will be given in the following.

6.2.1 Key Scheduling Function

Orthros adopts two bit-permutation-based key scheduling functions called KSF1 and KSF2, which are used to generate RK_r^1 and RK_r^2 ($0 \le r \le 12$) from the same 128-bit key K for Branch1 and Branch2, respectively. The whitening keys are RK_0^1 and RK_0^2, which will be first XORed with X^1 and X^2, respectively. RK_r^1 and RK_r^2 ($1 \le r \le 12$) are the round keys used in the r-th round of Branch1 and Branch2, respectively. The algorithms of KSF1 and KSF2 are shown in Fig 6.2. The bit-permutation P_{bk1} and P_{bk2} used in KSF1 and KSF2 are shown in Table 6.3.

In Fig 6.2, when K and RK_r^j are expressed in bit level, we have $RK_r^j = (rk_{r,0}^j, rk_{r,1}^j \cdots rk_{r,127}^j)$ and $K = (k_0, k_1 \cdots k_{127})$, where $rk_{r,i}^j, k_i \in F_2$ ($0 \le i \le 127, j \in \{1, 2\}$). In addition, $rk_{r,0}^j$ and k_0 are the most significant bit of RK_r^j and K, respectively.

[11] `https://github.com/subhadeep-banik/orthros`

[12] For convenience, we may call an input (an output) of Orthros a plaintext (a ciphertext), although it is not a block cipher.

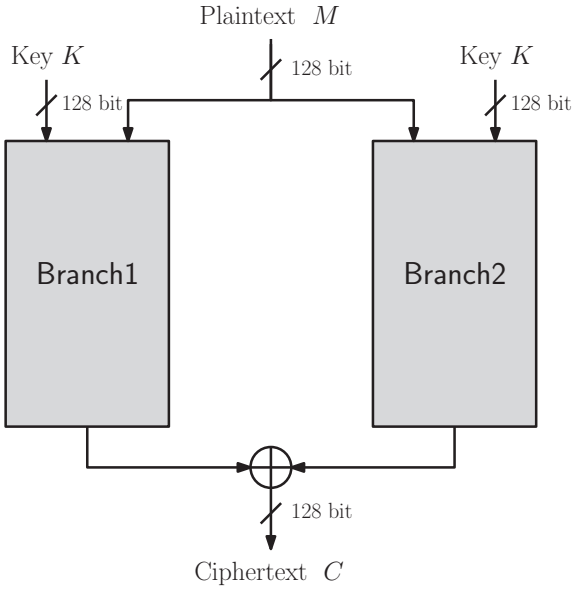


Figure 6.1: Overview of Orthros.

6.2.2 Round Function of Branch1 and Branch2

In this section, we present the details of Branch1 and Branch2, each of which is a 128-bit keyed permutation consisting of 12 rounds. The round keys (and whitening keys) (RK_r^1, RK_r^2) are first generated by KSF1 and KSF2. After adding the whitening keys, the 128-bit input will be processed via Branch1 and Branch2 as follows.

For the first 4 rounds of Branch1 and Branch2, the round function R is described as

$$R = \texttt{AddConstant} \circ \texttt{AddRoundKey} \circ \texttt{matrixMul} \circ \texttt{bit-permutation} \circ \texttt{S-box},$$

where `AddConstant` and `AddRoundKey` represent the constant addition operation and the round key addition operation, respectively.

For the following 7 rounds, the round function R' is described as

$$R' = \texttt{AddConstant} \circ \texttt{AddRoundKey} \circ \texttt{matrixMul} \circ \texttt{nibble-permutation} \circ \texttt{S-box}.$$

The sequence of operations in the last round is `AddConstant` $\circ$ `AddRoundKey` $\circ$ `S-box`. As described, the 128-bit outputs of Branch1 and Branch2 are XORed to generate the 128-bit ciphertext. Each component in the round function of Branch1 and Branch2 is described as follows.

S-box (`S-box`). A 4-bit S-box will be applied to each nibbles in parallel for Branch1 and Branch2. The specification of the 4-bit S-box is displayed in Table 6.1.

Table 6.1: S-box in Branch1 and Branch2.

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

Permutation (`bit-permutation`, `nibble-permutation`). For the first 4 rounds of Branch1 and Branch2, P_{br1} and P_{br2} will be used respectively. From the 5th round to the 11th round, the nibble permutations P_{n1} and P_{n2} will be adopted in each branch respectively. The details of the permutation P_{brN} and P_{nN}, where $N \in \{1, 2\}$, are shown in Table 6.4 and Table 6.5, respectively.

Algorithm KSF1(K)

1. $(k_0 \| k_1 \| \cdots \| k_{127}) \leftarrow K$
2. **for** $r = 0$ **to** 12 **do**
3. $\quad (rk^1_{r,0} \| rk^1_{r,1} \| \cdots \| rk^1_{r,127}) \leftarrow RK^1_r$
4. $\quad$ **if** $r = 0$ **then**
5. $\quad\quad$ **for** $i = 0$ **to** 127 **do**
6. $\quad\quad\quad rk^1_{0,P_{bk1}(i)} \leftarrow k_i$
7. $\quad\quad$ **end for**
8. $\quad$ **else**
9. $\quad\quad rk^1_{r,P_{bk1}(i)} \leftarrow rk^1_{r-1,i}$
10. $\quad$ **end if**
11. **end for**
12. **for** $r = 0$ **to** 12 **do**
13. $\quad RK^1_r \leftarrow (rk^1_{r,0} \| rk^1_{r,1} \| \cdots \| rk^1_{r,127})$
14. **end for**
15. **return** $(RK^1_0, RK^1_1, \cdots RK^1_{12})$

Algorithm KSF2(K)

1. $(k_0 \| k_1 \| \cdots \| k_{127}) \leftarrow K$
2. **for** $r = 0$ **to** 12 **do**
3. $\quad (rk^2_{r,0} \| rk^2_{r,1} \| \cdots \| rk^2_{r,127}) \leftarrow RK^2_r$
4. $\quad$ **if** $r = 0$ **then**
5. $\quad\quad$ **for** $i = 0$ **to** 127 **do**
6. $\quad\quad\quad rk^2_{0,P_{bk2}(i)} \leftarrow k_i$
7. $\quad\quad$ **end for**
8. $\quad$ **else**
9. $\quad\quad rk^2_{r,P_{bk2}(i)} \leftarrow rk^2_{r-1,i}$
10. $\quad$ **end if**
11. **end for**
12. **for** $r = 0$ **to** 12 **do**
13. $\quad RK^2_r \leftarrow (rk^2_{r,0} \| rk^2_{r,1} \| \cdots \| rk^2_{r,127})$
14. **end for**
15. **return** $(RK^2_0, RK^2_1, \cdots RK^2_{12})$

Figure 6.2: Algorithms of KSF1 and KSF2.

Matrix Multiplication (`matrixMul`). Let $\boldsymbol{M_b}$ be 4×4 matrix over nibbles defined as

$$\boldsymbol{M_b} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}.$$

Four nibbles (a_0, a_1, a_2, a_3) will be updated as follows:

$$(a_0, a_1, a_2, a_3)^T \leftarrow \boldsymbol{M_b} \cdot (a_0, a_1, a_2, a_3)^T,$$

where $(a_0, a_1, a_2, a_3)^T$ denotes a transposition. Specifically,

$$\begin{aligned} a_0 &= a_1 \oplus a_2 \oplus a_3, \\ a_1 &= a_0 \oplus a_2 \oplus a_3, \\ a_2 &= a_0 \oplus a_1 \oplus a_3, \\ a_3 &= a_0 \oplus a_1 \oplus a_2. \end{aligned}$$

AddRoundKey (`AddRoundKey`). In the r-th round, the internal states in Branch1 and Branch2 will be xored with the corresponding round key RK^1_r and RK^2_r, respectively.

AddConstant (`AddConstant`). The internal state will be xored with the corresponding round constant in each branch. The specification of the round constants is displayed in Table 6.2, where RC^1_r and RC^2_r represent the round constant used in the r-th round of Branch1 and Branch2, respectively.

6.2.2.1 Round Constants. In a similar manner to PRINCE, the round constants are derived from the fraction part of $\pi = 3.1415926....$ Specifically, if the fraction part of $\pi = 3.1415926...$ is expressed in binary string, it will be `00010100000101011001`.... By shifting left the binary string by 3 bits, we obtain a binary string `1010 0000 1010 1100 1`..., which corresponds to a nibble string `0xa`, `0x0`, `0xa`, `0xc`, and so on.

Table 6.2: Specification of the round constants RC_r^1 and RC_r^2.

Branch1	
Round Number r	RC_r^1
1	0xa0ac9329ac4bc991c2313219c193ca81
2	0x4420cb8b49cc9ba882c104ba4a22c918
3	0x3c0b2031431044cc31401a4129a108b8
4	0x33cc10a4043289941183323849c22304
5	0xaa82c1118b929aca0409424088ba2814
6	0x2081380c9c290882aacb223114a44aa4
7	0x981c0cb22144084bab32c99a2309423a
8	0xb24119bc33c18b2938900c848a2b242b
9	0x3491a301a430822a1933241099c9b039
10	0x301248a0939b922c380330318aac40ba
11	0x440a904904b141492a048b8a9b21b3c4
12	0x92c81b00089982982a44102332909c20

Branch2	
Round Number r	RC_r^2
1	0xa34a8ca0a88b04a1982b9381b2bacac8
2	0xca98490c308b9c0c99308bc988288c2a
3	0x403a2311bccb13a4ab39a8c42ba93924
4	0x48913c9c0c1808ca4894c19b399b1220
5	0x32b3218430109ca4a31ca91239b8c838
6	0x10bcc304a1b813b829c90b8bb1498bb3
7	0xa91c233a40c233b34a028990002b4093
8	0x8a2931ab0413bc2bb89a13abbc4b048b
9	0x9b1b8bc390a342204809124a9a180a32
10	0xa4ac29b88283c913cb4492c491aa100c
11	0xcab089094810cb043201a20c0acc09b1
12	0x4bba3b8984cb028c3839089a4cccccc1

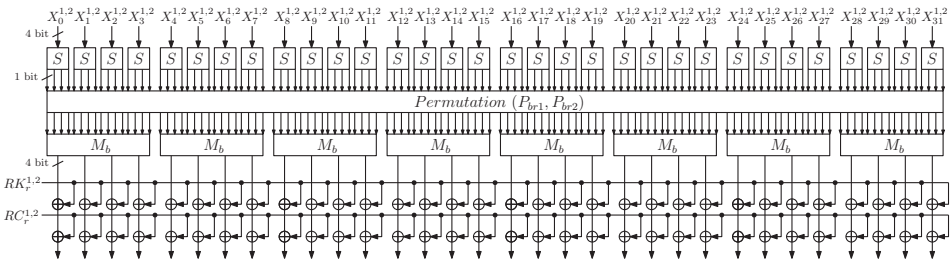


Figure 6.3: The round function of Branch1 and Branch2 in the first 4 rounds.

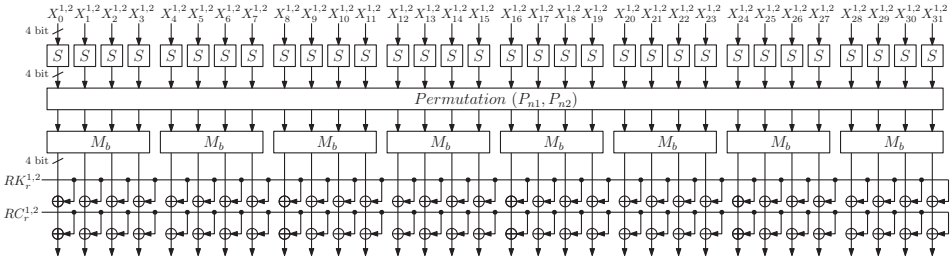


Figure 6.4: The round function of Branch1 and Branch2 in the last 8 rounds. The nibble permutation and the matrix multiplication in the last round will be omitted.

6.2.2.2 Illustration of Round Function. The illustration of the round function in the first 4 rounds of Branch1 and Branch2 is shown in Fig 6.3. The illustration of the round function in the last 8 rounds is shown in Fig 6.4.

6.2.2.3 Pseudocode. The algorithms of Branch1 and Branch2 are shown in Fig 6.5. In the pseudocode, when the 128-bit internal state is expressed in bits, we have $X = (x_0, x_1, \cdots x_{127})$ and x_0 is the most significant bit of X. When the 128-bit internal state is expressed in nibbles, we have $X = (X_0, X_1, \ldots, X_{31})$ and X_0 is the most significant nibble of X. For the constant addition, the 128-bit round constant RC_r^j is expressed in nibbles as $RC_r^j = (RC_{r,0}^j, RC_{r,1}^j, \cdots, RC_{r,31}^j)$, for $1 \leq r \leq 12$ and $1 \leq j \leq 2$, where j denotes the index of Branch. Similarly for the round-key addition, the 128-bit round key RK_r^j is expressed in nibbles as $RK_r^j = (RK_{r,0}^j, RK_{r,1}^j, \ldots, RK_{r,31}^j)$, for $1 \leq r \leq 12$ and $1 \leq j \leq 2$.

6.2.2.4 Processing. The 128-bit ciphertext C is generated by XORing the output of Branch1 and Branch2. The processing algorithm of Orthros is shown in Fig 6.6.

6.2.2.5 Claimed Security. Orthros claims single-key security, and does not claim any security in related-key and known/chosen-key settings.

6.2.3 Test Vectors

Table 6.6 presents two test vectors for Orthros.

6.3 Design Rationale

6.3.1 General Construction

As described in introduction, the overall structure of Orthros is a sum of two keyed permutations. This structure and its variant has been extensively studied in the literature [31, 59, 70, 99, 121, 141]. In particular, if two keyed permutations of Orthros (Branch1 and Branch2) were independent PRPs, we could claim n-bit provable security – more specifically the PRF advantage of $(q/2^n)^{1.5} + 2\epsilon(q, t + O(q))$ for $n = 128$ and q adaptive queries and time complexity t, where $\epsilon(q, t)$ denotes the PRP advantages of Branch1 and Branch2 with q queries and t time [70]. However, this means that either Branch1 or Branch2 could be already usable as a low-latency PRP, implying that they should have a sufficient amount of security margins against known cryptanalysis. Since each Branch never gives its outputs in clear, we expect that a pair of weak permutations can suffice to have a desired, n-bit secure PRF. Generally this approach is described as "prove-then-prune" [93]. This means that the provable security reduction does not hold anymore, and an implication of the

Algorithm $\mathsf{Branch}N(K, X)$

1. $(RK_0^N \parallel RK_1^N \parallel \ldots \parallel RK_{12}^N) \leftarrow \mathsf{KSF}N(K)$
2. $X \leftarrow X \oplus RK_0^N$
3. $(X_0 \parallel X_1 \parallel \ldots \parallel X_{31}) \leftarrow X$
4. **for** $r = 1$ **to** 11 **do**
5. $\quad (RC_{r,0}^N \parallel RC_{r,1}^N \parallel \ldots \parallel RC_{r,31}^N) \leftarrow RC_r^N$
6. $\quad (RK_{r,0}^N \parallel RK_{r,1}^N \parallel \ldots \parallel RK_{r,31}^N) \leftarrow RK_r^N$
7. $\quad$ **for** $i = 0$ **to** 31 **do**
8. $\quad\quad X_i \leftarrow S(X_i)$
9. $\quad$ **end for**
10. $\quad$ **if** $r < 5$ **then**
11. $\quad\quad$ **for** $i = 0$ **to** 31 **do**
12. $\quad\quad\quad (x_{4i} \parallel x_{4i+1} \parallel x_{4i+2} \parallel x_{4i+3}) \leftarrow X_i$
13. $\quad\quad$ **end for**
14. $\quad\quad (x'_0, x'_1, \ldots, x'_{127}) \leftarrow (x_0, x_1, \ldots, x_{127})$
15. $\quad\quad$ **for** $i = 0$ **to** 127 **do**
16. $\quad\quad\quad x_{P_{brN}(i)} \leftarrow x'_i$
17. $\quad\quad$ **end for**
18. $\quad\quad$ **for** $i = 0$ **to** 31 **do**
19. $\quad\quad\quad X_i \leftarrow (x_{4i} \parallel x_{4i+1} \parallel x_{4i+2} \parallel x_{4i+3})$
20. $\quad\quad$ **end for**
21. $\quad$ **else**
22. $\quad\quad (X'_0, X'_1, \ldots, X'_{31}) \leftarrow (X_0, X_1, \ldots, X_{31})$
23. $\quad\quad$ **for** $i = 0$ **to** 31 **do**
24. $\quad\quad\quad X_{P_{nN}(i)} \leftarrow X'_i$
25. $\quad\quad$ **end for**
26. $\quad$ **end if**
27. $\quad$ **for** $i = 0$ **to** 7 **do**
28. $\quad\quad (X_{4i}, X_{4i+1}, X_{4i+2}, X_{4i+3})^T \leftarrow \boldsymbol{M_b} \cdot (X_{4i}, X_{4i+1}, X_{4i+2}, X_{4i+3})^T$
29. $\quad$ **end for**
30. $\quad$ **for** $i = 0$ **to** 31 **do**
31. $\quad\quad X_i \leftarrow X_i \oplus RK_{r,i}^N \oplus RC_{r,i}^N$
32. $\quad$ **end for**
33. **end for**
34. **for** $i = 0$ **to** 31 **do**
35. $\quad X_i \leftarrow S(X_i)$
36. **end for**
37. $(RC_{12,0}^N \parallel RC_{12,1}^N \parallel \ldots \parallel RC_{12,31}^N) \leftarrow RC_{12}^N$
38. $(RK_{12,0}^N \parallel RK_{12,1}^N \parallel \ldots \parallel RK_{12,31}^N) \leftarrow RK_{12}^N$
39. **for** $i = 0$ **to** 31 **do**
40. $\quad X_i \leftarrow X_i \oplus RK_{12,i}^N \oplus RC_{12,i}^N$
41. **end for**
42. $Y \leftarrow (X_0 \parallel X_1 \parallel \ldots \parallel X_{31})$
43. **return** Y

Figure 6.5: Algorithms of Branch1 and Branch2, where $N \in \{1, 2\}$.

Algorithm $\mathsf{Orthros}(K, M)$

1. $X^1 \leftarrow M$, $X^2 \leftarrow M$
2. $Y^1 \leftarrow \mathsf{Branch1}(K, X^1)$, $Y^2 \leftarrow \mathsf{Branch2}(K, X^2)$
3. $C \leftarrow Y^1 \oplus Y^2$
4. **return** C

Figure 6.6: Processing algorithm of Orthros.

Table 6.3: Bit permutation P_{bkN} for key scheduling $\mathsf{KSF}N$, where $N \in \{1, 2\}$.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{bk1}(x)$	0	53	87	73	22	95	99	48	61	36	108	1	24	67	119	93
$P_{bk2}(x)$	76	30	53	35	31	46	2	79	11	125	110	87	39	91	14	101
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_{bk1}(x)$	54	103	69	112	16	111	94	122	31	66	33	83	47	3	65	62
$P_{bk2}(x)$	97	118	36	48	29	80	57	115	49	18	74	85	61	82	105	126
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P_{bk1}(x)$	123	9	101	19	5	58	89	37	38	51	28	106	82	76	121	4
$P_{bk2}(x)$	70	12	47	111	51	17	66	1	60	96	116	71	81	114	104	15
x	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P_{bk1}(x)$	70	7	42	92	104	80	45	75	114	17	2	97	46	107	63	18
$P_{bk2}(x)$	42	124	100	4	113	44	75	89	23	0	84	107	32	26	88	8
x	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
$P_{bk1}(x)$	109	15	127	43	13	59	29	125	77	11	50	30	12	90	118	64
$P_{bk2}(x)$	69	121	38	94	37	86	54	21	62	123	41	10	16	95	117	65
x	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
$P_{bk1}(x)$	20	35	57	10	124	56	68	91	116	21	84	98	52	81	126	34
$P_{bk2}(x)$	45	50	72	20	109	58	7	67	108	28	3	55	92	103	24	5
x	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
$P_{bk1}(x)$	105	27	120	74	6	85	40	72	113	41	23	49	79	55	102	8
$P_{bk2}(x)$	77	9	27	102	122	6	106	22	99	34	90	56	43	83	120	64
x	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
$P_{bk1}(x)$	117	39	88	26	25	110	14	32	115	100	86	71	78	44	96	60
$P_{bk2}(x)$	78	59	119	93	40	98	52	68	112	33	63	25	19	73	127	13

Table 6.4: Bit permutation P_{brN} for round function $\mathsf{Branch}N$, where $N \in \{1, 2\}$.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{br1}(x)$	6	46	62	126	70	52	28	14	36	125	72	83	106	95	4	35
$P_{br2}(x)$	20	122	74	62	119	35	15	66	9	85	32	117	21	83	127	106
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_{br1}(x)$	25	41	10	76	87	74	120	42	88	21	11	67	64	38	112	50
$P_{br2}(x)$	11	98	115	59	71	90	56	26	2	44	103	121	114	107	68	16
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$P_{br1}(x)$	85	109	24	65	99	0	49	37	8	66	114	47	127	100	56	40
$P_{br2}(x)$	84	1	102	33	80	52	76	36	27	94	37	55	82	12	112	64
x	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$P_{br1}(x)$	13	117	78	86	92	58	124	101	55	89	97	9	18	116	59	15
$P_{br2}(x)$	105	14	91	17	108	124	6	93	29	86	123	79	72	53	19	99
x	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
$P_{br1}(x)$	20	45	75	2	77	27	1	60	115	107	26	69	119	3	84	51
$P_{br2}(x)$	50	18	81	73	67	88	4	61	111	49	24	45	57	78	100	22
x	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
$P_{br1}(x)$	123	110	31	82	113	53	81	102	63	118	93	12	30	94	108	32
$P_{br2}(x)$	110	47	116	54	60	70	97	39	3	41	48	96	23	42	113	87
x	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
$P_{br1}(x)$	5	111	29	43	91	19	79	33	73	44	98	48	22	61	68	105
$P_{br2}(x)$	126	13	31	40	51	25	65	125	8	101	118	28	38	89	5	104
x	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
$P_{br1}(x)$	34	71	54	104	17	57	80	103	96	121	23	39	122	90	7	16
$P_{br2}(x)$	109	120	69	43	7	77	58	34	10	63	30	95	75	46	0	92

Table 6.5: Nibble permutation P_{nN} for round function $\mathsf{Branch}N$, where $N \in \{1, 2\}$.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{n1}(x)$	10	27	5	1	30	23	16	13	21	31	6	14	0	25	11	18
$P_{n2}(x)$	26	13	7	11	29	0	17	21	23	5	18	25	12	10	28	2
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$P_{n1}(x)$	15	28	19	24	7	8	22	3	4	29	9	2	26	20	12	17
$P_{n2}(x)$	14	19	24	22	1	8	4	31	15	6	27	9	16	30	20	3

Table 6.6: Test vectors for Orthros in hex.

Plaintext	00000000000000000000000000000000
Key	00000000000000000000000000000000
Ciphertext	6060acb118f411e434ba4e01984de0de
Plaintext	a947436710924ccd47f2d571deea8f05
Key	4a2be60e3db6abe0c03eaec66fd05d0c
Ciphertext	e4cec0d077a3401d8c4d07b6d5196e5f

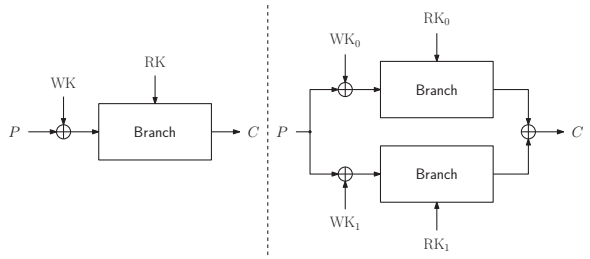


Figure 6.7: (Left) The toy cipher using a single branch and (Right) that using double branches, where WK and RK are the whitening key and round key, respectively.

security bound is more or less fuzzy and depends on the scheme. In our analysis we did not find a full-round attack against Branch1 and Branch2 as a PRP, however they have rather slim margins as a standalone block cipher. Therefore, we bolster our security claim with an extensive security analysis on the whole construction.

We note that realizing Branch1 and Branch2 as Even-Mansour ciphers [82] can reduce the circuit size thanks to the absence of key schedule. However, it can provide $2n/3$-bit PRF security at best, as proved by Chen et al. [59]. Instead, we adopt bit permutation-based key scheduling functions for its hardware friendliness.

In order to investigate the initial security of the sum of permutations from the perspective of cryptanalysis, we compare in the next section the differential and linear behaviour for two toy ciphers adopting single branch and double branches, respectively. However, it should be emphasized that the security of Orthros never relies on our experiments on the toy ciphers but rather a comprehensive study of Orthros, as will be detailed in Section 6.4.

6.3.2 Toy Ciphers

The unique feature of our design is the use of two parallel branches (effectively block ciphers). In order to investigate the generic security of this design, we introduce two toy ciphers using single branch and double branches as shown in Fig. 6.7. We focus on the maximal differential probability (MDP) [44] and linear bias [124] as they are two of the most fundamental security metrics. For the underlying branches, we consider both SPN and Feistel structures.

6.3.2.1 Experiments for a SPN-based toy cipher. First, we consider the case of SPN with 16-bit internal state and 16-bit round key. The basic design of this SPN-based toy cipher is similar to Midori. For our SPN-based toy cipher, the round function is composed of the following operations: S-box, Shuffle, Mix and AK. The state is organized as a 4×4 two-dimensional Boolean array A. The $(4j + i)$-th bit of the internal state is placed at $A[i][j]$ $(0 \leq i \leq 3, 0 \leq j \leq 3)$. For the S-box, each column $A[\cdot][j]$ $(0 \leq j \leq 3)$ is viewed as a 4-bit value

$x = 8 \cdot A[3][j] + 4 \cdot A[2][j] + 2 \cdot A[1][j] + A[0][j]$ and the internal state is updated as $A[\cdot][j]$=S-box$(A[\cdot][j])$. The S-box is the same with that used in Orthros. For the Shuffle operation, the state is reorganized as $A[i][j] = A[i'][j']$ $(0 \leq i \leq 3, 0 \leq j \leq 3)$, where $4j' + i' = \text{SF}[4j + i]$ and the array SF is defined as

$$\text{SF}[16] = [0, 10, 5, 15, 14, 4, 11, 1, 9, 3, 12, 6, 7, 13, 2, 8].$$

For the Mix operation, each column is updated by multiplying the binary matrix M_b, i.e., $(A[0][j], A[1][j], A[2][j], A[3][j])^T = M_b \cdot (A[0][j], A[1][j], A[2][j], A[3][j])^T$ $(0 \leq j \leq 3)$. For the AK operation, a random 16-bit round key will be XORed with the 16-bit internal state. Since the construction of the underlying block cipher used in Orthros is somewhat similar to that of Midori, we adopt the same Shuffle and Mix operations as in Midori for the toy cipher in order to construct a 16-bit toy cipher.

To compute MDP over a certain number of rounds for each construction, we first generate a random value for the round keys. Then, the whole block cipher can be viewed as a large 16-bit S-box. By exhausting all possible $2^{16} \times (2^{16} - 1)/2$ input pairs, we count the number of occurrences for each possible output difference and obtain the maximum frequency of the output difference, which we denote by CNT_0. In this way, MDP is calculated as $\text{CNT}_0/2^{16}$. With this method, we carried out 100 experiments and compute MDP for each experiments, and take the maximum for all experiments. The results are displayed in Table 6.7. Table 6.7 shows that the maximal value becomes stable after about 4 rounds in the construction using double branches. For the construction using a single branch, it becomes stable in about 5 rounds.

Table 6.7: The maximal differential probability of SPN-based toy ciphers.

Rounds \ Max Pro.	Single Branch	Double Branches
1	2^{-3}	2^{-1}
2	2^{-8}	2^{-6}
3	2^{-8}	$2^{-5.7}$
4	$2^{-11.3}$	$2^{-12.3}$
5	$2^{-12.5}$	$2^{-12.5}$
6	$2^{-12.5}$	$2^{-12.5}$
7	$2^{-12.5}$	$2^{-12.5}$
8	$2^{-12.5}$	$2^{-12.5}$

6.3.2.2 Experiments for a Feistel-based toy cipher. For the case of Feistel-based cipher, we consider 4-GFS (generalized Feistel structure with 4 sub-blocks), as shown in Fig. 6.8. The round function consists of two parallel 4-bit S-boxes with random round keys, where the S-box is the same with that used in Orthros. We carried out 100 experiments and compute the maximum of the MDP for all experiments, for both the constructions using a single branch and double branches. The corresponding results are displayed in Table 6.8. It shows that the maximal value becomes stable after about 7 rounds in the construction using double branches. For the construction using single branch, it becomes stable in about 10 rounds.

6.3.2.3 Experiments on linear masks. Similar experiments have also been performed to evaluate the maximal linear bias for the toy ciphers. Due to the high time complexity to accurately compute the maximal linear bias, we turn to calculating it in a probabilistic way. Specifically, we

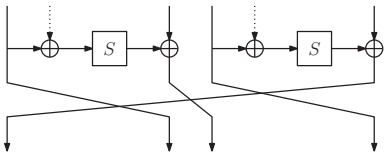


Figure 6.8: 4-GFS. Dotted lines denote (random) round keys.

Table 6.8: The maximal differential probability for each GFS-based construction.

Rounds \ Max Pro.	Single Branch	Double Branches
1	2^{-1}	2^{-1}
2	2^{-3}	2^{-1}
3	2^{-5}	2^{-1}
4	2^{-7}	$2^{-5.4}$
5	2^{-7}	$2^{-7.3}$
6	$2^{-9.6}$	$2^{-11.1}$
7	$2^{-9.4}$	$2^{-12.5}$
8	2^{-11}	$2^{-12.5}$
9	2^{-12}	$2^{-12.5}$
10	$2^{-12.5}$	$2^{-12.5}$
11	$2^{-12.5}$	$2^{-12.5}$
12	$2^{-12.5}$	$2^{-12.5}$

randomly choose some input and output masks and select the maximal linear bias from them. For the SPN-based toy cipher, it is found that the maximal linear bias $(2^{-6.4})$ becomes stable after 4 rounds if using a single branch, while it becomes stable in 3 rounds for double branches. For the GFS-based toy cipher, the maximal linear bias $(2^{-6.4})$ becomes stable in 9 rounds and 6 rounds for a single branch and double branches, respectively.

6.3.2.4 Summary. In our experiments, both the maximal differential probability and linear bias of the double branches reach a stable value in a smaller number of rounds than that of the single branch. Of course the scale of experiment is limited and a more theoretical support should be desired. However, these results suggest that the double branch enhances the security of the single branch. We also emphasize that the security of Orthros is never ensured based on such a simple simulation. Instead, a comprehensive study is performed.

6.3.3 Linear Layer

As underlying matrices in the linear layer, we adopt 4×4 almost MDS binary matrix used in Midori [11], whose delay is much smaller than MDS matrices. However, as discussed in [11], its diffusion speed is slower and the lower bounds of the number of active S-boxes in each round is smaller than those of ciphers employing MDS matrices due to its lower branch number. To improve the diffusion speed and to increase active S-boxes in each round, we utilize bit and nibble permutations in a hybrid manner. We will see that this enables to guarantee security with a relatively small number of rounds.

Midori-128 [11] also adopted a bit and a nibble permutations, however, our design is more efficient in term of the diffusion speed and the number of active S-boxes while keeping the same hardware cost in an unrolled implementation. Specifically, we adopt two different linear layers that consists of a bit permutation and a nibble permutation, i.e., a bit permutation is used for some rounds and a nibble permutation for the rest, while Midori-128 [11] uses a single linear layer including both of a bit permutation and a nibble permutation. Importantly, this change of linear layers does not require any additional hardware cost in an unrolled implementation, as observed by [53].

6.3.3.1 Bit Permutation vs Nibble Permutation. To see the advantage of our hybrid use of bit and nibble permutations over the consistent use of a bit or a nibble permutation, we compare the diffusion effect and the lower bound of the number of active S-boxes of two different 128-bit block SPN structures, which we call SPN-B and SPN-N. Here, SPN-B (SPN-N) consistently uses a bit (nibble) permutation. Both use the same 4-bit S-box and the matrix as those used in a branch of Orthros. In addition, we used the full-diffusion property of S-box when evaluating the diffusion effect.

6.3.3.2 Diffusion. To evaluate the minimum number of rounds that achieves the full diffusion for each SPN-B and SPN-N, we look into the propagation of one active input bit and count the upper bounds of the number of active bits through each operation. Here, we only need to consider the number of active bits after `S-box` and `matrixMul` as the remaining operations do not affect its value. The upper bounds of the number of active bits after each operation over some rounds are shown in Table 6.9.

According to Table 6.9, SPN-B and SPN-N require at least 2.5 rounds (2 rounds plus `S-box`) and 4 rounds, respectively, i.e., the *optimal* numbers of rounds for the full diffusion of SPN-B and SPN-N are estimated as 2.5 and 4 rounds, respectively. We conclude that there is a clear gap between bit permutations and nibble permutations in terms of the diffusion. Note that a class of

Table 6.9: The upper bound of the number of active bits after each operation.

Round	Operation	Structures	
		SPN-B	SPN-N
-	Input	1	1
1	S-box	4	4
	matrixMul	12	12
2	S-box	48	12
	matrixMul	120	36
3	S-box	**128**	36
	matrixMul	128	100
4	S-box	128	100
	matrixMul	128	**128**

bit permutations covers all nibble permutations. Thus, to achieve the 2.5-round full diffusion, we need to find a class of bit permutations that are not included in a class of nibble permutations.

6.3.3.3 Active S-box. Mixed Integer Linear Programming (MILP) is used to obtain the lower bound of the number of active S-boxes in each round. Unfortunately, it is computationally infeasible to estimate the lower bound of the number of active S-boxes of SPN-B, except for a very small number of rounds, due to the large search space of 128-bit bitwise differential and linear trails. This problem about the use of bit permutation was also pointed out by the designers of QARMA-128 [9]. As a consequence, QARMA-128 does not claim a lower bound of active S-boxes. Indeed, it was infeasible to compute a lower bound of the number of active S-boxes for more than 5 rounds for SPN-B, even with a computer equipped with 48 cores and 256 GB RAM. Therefore, SPN-B can only guarantee a very small number of active S-boxes for a moderately large number of rounds (say 10), since we have to combine the bounds obtained for a small number of rounds, which generally yields a loose bound. For example, the lower bound for 8 rounds is obtained by the sum of the bounds for 4 rounds. In our experiment, the best lower bound for 4 rounds is 16, therefore the obtained bound for 8 rounds is only 32.

On the other hand, for SPN-N, it is feasible to obtain a tight lower bound of the number of active S-boxes up to 8 rounds, using the aforementioned 48-core computer. The evaluation of one candidate requires about 2 days by the same computer. As a result, we found a class of nibble permutations that attain 60 active S-boxes over 8 rounds. We will explain the details in the following section.

6.3.3.4 Conclusion. Based on the above discussions, we conclude that a structure employing a single permutation cannot achieve a fast full diffusion and a guaranteed large number of active S-boxes simultaneously. Hence, we decided to use a bit permutation for the first few rounds, and use a nibble permutation for the rest of the rounds. Consequently, Orthros reaches the full diffusion after first 2.5 rounds and guarantees more than 64 active S-boxes over 10 rounds.

6.3.3.5 Finding Optimal Bit Permutations for Diffusion. We take a two-step approach to find a class of bit permutations that achieves 2.5-round full diffusion for SPN-B, which turns out to be optimal. Let S denote the 128-bit internal state S of SPN-B. It is also

viewed as a 4×8 two-dimensional nibble array:

$$S = \begin{bmatrix} S_0 & S_4 & S_8 & S_{12} & S_{16} & S_{20} & S_{24} & S_{28} \\ S_1 & S_5 & S_9 & S_{13} & S_{17} & S_{21} & S_{25} & S_{29} \\ S_2 & S_6 & S_{10} & S_{14} & S_{18} & S_{22} & S_{26} & S_{30} \\ S_3 & S_7 & S_{11} & S_{15} & S_{19} & S_{23} & S_{27} & S_{31} \end{bmatrix},$$

where S_i $(0 \leq i \leq 31)$ is a 4-bit value defined as

$$S_i = [s_{4i}, s_{4i+1}, s_{4i+2}, s_{4i+3}]^T.$$

Note that S can also be viewed as a 16×8 two-dimensional binary array by seeing each column of S as a 16-bit sequence. Hereafter, *bit-cell* means a binary cell in the 16×8 array and *nibble-cell* means a nibble cell in the 4×8 array, that is, S_i.

We first try to reduce the search space of target 128-bit permutations as it is computationally infeasible to test all possible $2^{716.16}(= 128!)$ candidates. Specifically, we focus on a class satisfying Condition 1 to efficiently find the class of bit permutations having the 2.5 rounds full diffusion property.

Condition 1 *For any nibble-cell S_i $(0 \leq i \leq 31)$, the corresponding 4 bits, $s_{4i}, s_{4i+1}, s_{4i+2}, s_{4i+3}$, are mapped to the bit-cells in different columns after applying the bit permutation.*

Detailed description of Condition 1 Fig 6.9 shows the transition of the state after applying P_{bk1}, which satisfies Condition 1. The state is represented as a 16×8 bit array, where S_0, S_1, S_2, S_3 are nibbles consisting of the first state column. Let us focus on the leftmost column. After applying a bit-permutation satisfying Condition 1, for $0 \leq i \leq 3$, Fig 6.9 shows that the 4 bits of S_i are mapped to different columns. The same applies to the remaining 7 columns.

$$\begin{bmatrix} s_0 & s_{16} & s_{32} & s_{48} & s_{64} & s_{80} & s_{96} & s_{112} \\ s_1 & s_{17} & s_{33} & s_{49} & s_{65} & s_{81} & s_{97} & s_{113} \\ s_2 & s_{18} & s_{34} & s_{50} & s_{66} & s_{82} & s_{98} & s_{114} \\ s_3 & s_{19} & s_{35} & s_{51} & s_{67} & s_{83} & s_{99} & s_{115} \\ s_4 & s_{20} & s_{36} & s_{52} & s_{68} & s_{84} & s_{100} & s_{116} \\ s_5 & s_{21} & s_{37} & s_{53} & s_{69} & s_{85} & s_{101} & s_{117} \\ s_6 & s_{22} & s_{38} & s_{54} & s_{70} & s_{86} & s_{102} & s_{118} \\ s_7 & s_{23} & s_{39} & s_{55} & s_{71} & s_{87} & s_{103} & s_{119} \\ s_8 & s_{24} & s_{40} & s_{56} & s_{72} & s_{88} & s_{104} & s_{120} \\ s_9 & s_{25} & s_{41} & s_{57} & s_{73} & s_{89} & s_{105} & s_{121} \\ s_{10} & s_{26} & s_{42} & s_{58} & s_{74} & s_{90} & s_{106} & s_{122} \\ s_{11} & s_{27} & s_{43} & s_{59} & s_{75} & s_{91} & s_{107} & s_{123} \\ s_{12} & s_{28} & s_{44} & s_{60} & s_{76} & s_{92} & s_{108} & s_{124} \\ s_{13} & s_{29} & s_{45} & s_{61} & s_{77} & s_{93} & s_{109} & s_{125} \\ s_{14} & s_{30} & s_{46} & s_{62} & s_{78} & s_{94} & s_{110} & s_{126} \\ s_{15} & s_{31} & s_{47} & s_{63} & s_{79} & s_{95} & s_{111} & s_{127} \end{bmatrix} \xRightarrow{P_{bk1}} \begin{bmatrix} s_0 & s_{20} & s_{119} & s_7 & s_{79} & s_{53} & s_{126} & s_{19} \\ s_{11} & s_{57} & s_{26} & s_{107} & s_{30} & s_{93} & s_{59} & s_{104} \\ s_{58} & s_{63} & s_{95} & s_{74} & s_{25} & s_{44} & s_{91} & s_{56} \\ s_{29} & s_{35} & s_{81} & s_{41} & s_{13} & s_{27} & s_6 & s_{120} \\ s_{47} & s_{80} & s_9 & s_{92} & s_{86} & s_{90} & s_{121} & s_{88} \\ s_{36} & s_{89} & s_{39} & s_1 & s_{18} & s_{101} & s_{34} & s_{112} \\ s_{100} & s_4 & s_{40} & s_{16} & s_{48} & s_{122} & s_{110} & s_{78} \\ s_{49} & s_{106} & s_{113} & s_{109} & s_{123} & s_2 & s_{17} & s_{14} \\ s_{111} & s_{12} & s_{102} & s_{85} & s_{103} & s_{114} & s_{52} & s_{98} \\ s_{33} & s_{116} & s_{105} & s_{82} & s_3 & s_{38} & s_{96} & s_{46} \\ s_{83} & s_{115} & s_{50} & s_{37} & s_{99} & s_{77} & s_{43} & s_{23} \\ s_{73} & s_{97} & s_{67} & s_{69} & s_{55} & s_{87} & s_{61} & s_{32} \\ s_{76} & s_{42} & s_{125} & s_{127} & s_{45} & s_{51} & s_{10} & s_{84} \\ s_{68} & s_{70} & s_{54} & s_8 & s_{72} & s_{15} & s_{64} & s_{71} \\ s_{118} & s_{75} & s_{60} & s_{31} & s_{124} & s_{22} & s_{117} & s_{94} \\ s_{65} & s_{24} & s_{28} & s_{62} & s_{108} & s_5 & s_{21} & s_{66} \end{bmatrix}$$

$$S_0 = (s_0, s_1, s_2, s_3)^T \quad S_1 = (s_4, s_5, s_6, s_7)^T \quad S_2 = (s_8, s_9, s_{10}, s_{11})^T \quad S_3 = (s_{12}, s_{13}, s_{14}, s_{15})^T$$

Figure 6.9: Transition of a state after applying P_{bk1}.

Condition 1 can be justified as follows. If a bit permutation satisfies Condition 1, for the 4 output bits of each S-box, they will be mapped to 4 different groups of inputs to the binary matrix. Observe that in the MatrixMul operation, the binary matrix M_b is independently applied to 8 different groups of inputs, each of which consists of 4 consecutive nibbles. This is expected to be

a key feature for the fast diffusion since 1 input active bit is expanded to 4 active bits via the S-box, and the 4 active bits are subsequently expanded to 12 active bits after the bit permutation and MatrixMul operations. This matches the upper bound of the number of active bits after one-round permutation, as shown in Table 6.9.

In the class of the bit permutations satisfying Condition 1, we obtain the following sufficient condition for the 2.5-round full diffusion. It should be emphasized that for each active bit in the nibble-cell S_i $(0 \leq i \leq 31)$, after applying the S-box and the bit permutation satisfying Condition 1, there will exist 4 different groups of inputs to the binary matrix, each of which will contain exactly one active nibble. Therefore, after matrixMul there will be 12 active nibbles. We add the following condition on these 12 active nibbles.

Condition 2 *After applying the bit permutation, in each column of the 4×8 array, there exist at least 2 nibble-cells containing the bits coming from those in the active 12 nibbles.*

Detailed description of Condition 2 For each active bit in the nibble cell S_i $(0 \leq i \leq 31)$ in the input, after applying the S-box and the bit permutation satisfying Condition 1, there will exist 4 different groups of inputs to the binary matrix, each of which will contain exactly one active nibble, as shown in Fig. 6.10. Therefore, there will be 12 active nibbles after MatrixMul in the first round, which will activate 12 nibbles located in the same positions in the second round after S-box operation, as depicted in Fig. 6.11. Therefore, there are 12 nibbles (48 bits in total) in the state after the S-box (with full-diffusion property) operation in the second round, independent of the value of the one active bit in the input to the first round. After applying a permutation satisfying Condition 2 for these 48 bits, in each column of the 4×8 array, there exist at least 2 nibble cells containing the bits coming from these 48 bits, as shown in Fig. 6.12 for bit level and in Fig. 6.13 for nibble level. In other words, after applying the bit permutation satisfying Condition 2 in the second round, in each column of the nibble array, there are at least 2 nibbles dependent of the one active bit in the input to the first round. When the MatrixMul operation is further applied to each column of the 4×8 nibble array, the values of all the four nibbles in each column will therefore dependent of the one active bit in the input to the first round. However, it cannot be guaranteed that the value of each bit will be dependent of the one active bit. Thus, after further applying the S-box with a full-diffusion property in the third round, all 128 bits become dependent of the one active bit. This means that the full diffusion is achieved by 2.5 rounds.

An example to explain the 2.5-round diffusion can be referred to Fig. 6.10, 6.11, 6.12, and Fig. 6.13.

The above description is sufficient for the 2.5 round full diffusion in the class of the bit permutations satisfying Condition 1.

Bit permutations of Table 6.4 used in Branch1 and Branch2 satisfy both Condition 1 and 2, respectively, i.e. attain 2.5-round full diffusion.

6.3.3.6 Finding Good Nibble Permutation for Active S-boxes. As in the case of bit permutations, we take the following two-step approach to find nibble permutations that can activate as many S-boxes as possible over a certain number of rounds, which is expected to outperform that used in Midori-128. To explain our approach, we use the same 4×8 array to express the 128-bit state as above.

In the first step, we reduce the search space by focusing on the class of nibble permutations satisfying Condition 3 since it is computationally infeasible to estimate the lower bound of the number of active S-boxes for all possible $2^{117.66}(= 32!)$ permutations. Condition 3 is chosen to achieve fast diffusion for differences and linear masks.

$$
\begin{bmatrix}
s_0 & s_{16} & s_{32} & s_{48} & s_{64} & s_{80} & s_{96} & s_{112} \\
s_1 & s_{17} & s_{33} & s_{49} & s_{65} & s_{81} & s_{97} & s_{113} \\
s_2 & s_{18} & s_{34} & s_{50} & s_{66} & s_{82} & s_{98} & s_{114} \\
s_3 & s_{19} & s_{35} & s_{51} & s_{67} & s_{83} & s_{99} & s_{115} \\
s_4 & s_{20} & s_{36} & s_{52} & s_{68} & s_{84} & s_{100} & s_{116} \\
s_5 & s_{21} & s_{37} & s_{53} & s_{69} & s_{85} & s_{101} & s_{117} \\
s_6 & s_{22} & s_{38} & s_{54} & s_{70} & s_{86} & s_{102} & s_{118} \\
s_7 & s_{23} & s_{39} & s_{55} & s_{71} & s_{87} & s_{103} & s_{119} \\
s_8 & s_{24} & s_{40} & s_{56} & s_{72} & s_{88} & s_{104} & s_{120} \\
s_9 & s_{25} & s_{41} & s_{57} & s_{73} & s_{89} & s_{105} & s_{121} \\
s_{10} & s_{26} & s_{42} & s_{58} & s_{74} & s_{90} & s_{106} & s_{122} \\
s_{11} & s_{27} & s_{43} & s_{59} & s_{75} & s_{91} & s_{107} & s_{123} \\
s_{12} & s_{28} & s_{44} & s_{60} & s_{76} & s_{92} & s_{108} & s_{124} \\
s_{13} & s_{29} & s_{45} & s_{61} & s_{77} & s_{93} & s_{109} & s_{125} \\
s_{14} & s_{30} & s_{46} & s_{62} & s_{78} & s_{94} & s_{110} & s_{126} \\
s_{15} & s_{31} & s_{47} & s_{63} & s_{79} & s_{95} & s_{111} & s_{127}
\end{bmatrix}
\xRightarrow{P_{bk,1}}
\begin{bmatrix}
s_0 & s_{20} & s_{119} & s_7 & s_{79} & s_{53} & s_{126} & s_{19} \\
s_{11} & s_{57} & s_{26} & s_{107} & s_{30} & s_{93} & s_{59} & s_{104} \\
s_{58} & s_{63} & s_{95} & s_{74} & s_{25} & s_{44} & s_{91} & s_{56} \\
s_{29} & s_{35} & s_{81} & s_{41} & s_{13} & s_{27} & s_6 & s_{120} \\
s_{47} & s_{80} & s_9 & s_{92} & s_{86} & s_{90} & s_{121} & s_{88} \\
s_{36} & s_{89} & s_{39} & s_1 & s_{18} & s_{101} & s_{34} & s_{112} \\
s_{100} & s_4 & s_{40} & s_{16} & s_{48} & s_{122} & s_{110} & s_{78} \\
s_{49} & s_{106} & s_{113} & s_{109} & s_{123} & s_2 & s_{17} & s_{14} \\
s_{111} & s_{12} & s_{102} & s_{85} & s_{103} & s_{114} & s_{52} & s_{98} \\
s_{33} & s_{116} & s_{105} & s_{82} & s_3 & s_{38} & s_{96} & s_{46} \\
s_{83} & s_{115} & s_{50} & s_{37} & s_{99} & s_{77} & s_{43} & s_{23} \\
s_{73} & s_{97} & s_{67} & s_{69} & s_{55} & s_{87} & s_{61} & s_{32} \\
s_{76} & s_{42} & s_{125} & s_{127} & s_{45} & s_{51} & s_{10} & s_{84} \\
s_{68} & s_{70} & s_{54} & s_8 & s_{72} & s_{15} & s_{64} & s_{71} \\
s_{118} & s_{75} & s_{60} & s_{31} & s_{124} & s_{22} & s_{117} & s_{94} \\
s_{65} & s_{24} & s_{28} & s_{62} & s_{108} & s_5 & s_{21} & s_{66}
\end{bmatrix}
$$

$S_0 = (s_0, s_1, s_2, s_3)^T$

Figure 6.10: The 4 active bits after the bit permutation in the first round, as marked in red.

$$
\begin{bmatrix}
S_0 & S_4 & S_8 & S_{12} & S_{16} & S_{20} & S_{24} & S_{28} \\
S_1 & S_5 & S_9 & S_{13} & S_{17} & S_{21} & S_{25} & S_{29} \\
S_2 & S_6 & S_{10} & S_{14} & S_{18} & S_{22} & S_{26} & S_{30} \\
S_3 & S_7 & S_{11} & S_{15} & S_{19} & S_{23} & S_{27} & S_{31}
\end{bmatrix}
$$

Figure 6.11: The 12 active nibbles after MatrixMul in the first round, marked in red.

$$
\begin{bmatrix}
s_0 & s_{16} & s_{32} & s_{48} & s_{64} & s_{80} & s_{96} & s_{112} \\
s_1 & s_{17} & s_{33} & s_{49} & s_{65} & s_{81} & s_{97} & s_{113} \\
s_2 & s_{18} & s_{34} & s_{50} & s_{66} & s_{82} & s_{98} & s_{114} \\
s_3 & s_{19} & s_{35} & s_{51} & s_{67} & s_{83} & s_{99} & s_{115} \\
s_4 & s_{20} & s_{36} & s_{52} & s_{68} & s_{84} & s_{100} & s_{116} \\
s_5 & s_{21} & s_{37} & s_{53} & s_{69} & s_{85} & s_{101} & s_{117} \\
s_6 & s_{22} & s_{38} & s_{54} & s_{70} & s_{86} & s_{102} & s_{118} \\
s_7 & s_{23} & s_{39} & s_{55} & s_{71} & s_{87} & s_{103} & s_{119} \\
s_8 & s_{24} & s_{40} & s_{56} & s_{72} & s_{88} & s_{104} & s_{120} \\
s_9 & s_{25} & s_{41} & s_{57} & s_{73} & s_{89} & s_{105} & s_{121} \\
s_{10} & s_{26} & s_{42} & s_{58} & s_{74} & s_{90} & s_{106} & s_{122} \\
s_{11} & s_{27} & s_{43} & s_{59} & s_{75} & s_{91} & s_{107} & s_{123} \\
s_{12} & s_{28} & s_{44} & s_{60} & s_{76} & s_{92} & s_{108} & s_{124} \\
s_{13} & s_{29} & s_{45} & s_{61} & s_{77} & s_{93} & s_{109} & s_{125} \\
s_{14} & s_{30} & s_{46} & s_{62} & s_{78} & s_{94} & s_{110} & s_{126} \\
s_{15} & s_{31} & s_{47} & s_{63} & s_{79} & s_{95} & s_{111} & s_{127}
\end{bmatrix}
\xRightarrow{P_{bk,1}}
\begin{bmatrix}
s_0 & s_{20} & s_{119} & s_7 & s_{79} & s_{53} & s_{126} & s_{19} \\
s_{11} & s_{57} & s_{26} & s_{107} & s_{30} & s_{93} & s_{59} & s_{104} \\
s_{58} & s_{63} & s_{95} & s_{74} & s_{25} & s_{44} & s_{91} & s_{56} \\
s_{29} & s_{35} & s_{81} & s_{41} & s_{13} & s_{27} & s_6 & s_{120} \\
s_{47} & s_{80} & s_9 & s_{92} & s_{86} & s_{90} & s_{121} & s_{88} \\
s_{36} & s_{89} & s_{39} & s_1 & s_{18} & s_{101} & s_{34} & s_{112} \\
s_{100} & s_4 & s_{40} & s_{16} & s_{48} & s_{122} & s_{110} & s_{78} \\
s_{49} & s_{106} & s_{113} & s_{109} & s_{123} & s_2 & s_{17} & s_{14} \\
s_{111} & s_{12} & s_{102} & s_{85} & s_{103} & s_{114} & s_{52} & s_{98} \\
s_{33} & s_{116} & s_{105} & s_{82} & s_3 & s_{38} & s_{96} & s_{46} \\
s_{83} & s_{115} & s_{50} & s_{37} & s_{99} & s_{77} & s_{43} & s_{23} \\
s_{73} & s_{97} & s_{67} & s_{69} & s_{55} & s_{87} & s_{61} & s_{32} \\
s_{76} & s_{42} & s_{125} & s_{127} & s_{45} & s_{51} & s_{10} & s_{84} \\
s_{68} & s_{70} & s_{54} & s_8 & s_{72} & s_{15} & s_{64} & s_{71} \\
s_{118} & s_{75} & s_{60} & s_{31} & s_{124} & s_{22} & s_{117} & s_{94} \\
s_{65} & s_{24} & s_{28} & s_{62} & s_{108} & s_5 & s_{21} & s_{66}
\end{bmatrix}
$$

Figure 6.12: The 48 active bits after bit permutation in the second round, marked in red.

$$
\begin{bmatrix}
S_0 & S_4 & S_8 & S_{12} & S_{16} & S_{20} & S_{24} & S_{28} \\
S_1 & S_5 & S_9 & S_{13} & S_{17} & S_{21} & S_{25} & S_{29} \\
S_2 & S_6 & S_{10} & S_{14} & S_{18} & S_{22} & S_{26} & S_{30} \\
S_3 & S_7 & S_{11} & S_{15} & S_{19} & S_{23} & S_{27} & S_{31}
\end{bmatrix}
$$

Figure 6.13: The active nibbles after bit permutation in the second round, marked in red.

Table 6.10: Comparison of lower bounds of the number of active S-boxes.

Target	4	5	6	7	8
Midori-128 [11]	16	20	30	35	38
Our nibble permutation	16	25	36	51	60

Condition 3 *For each column* $(S_{4i}, S_{4i+1}, S_{4i+2}, S_{4i+3})$ *in the* 4×8 *array, after applying the nibble permutation, they will be mapped to four nibble-cells in different columns.*

In the second step, we first randomly choose 7,000 nibble permutations satisfying Condition 3. Then, we compute the lower bound of the number of active S-boxes after 5, 6, 7 and 8 rounds for these nibble permutations. To efficiently find good permutations among these nibble permutations, we first find a class of nibble permutations that have the best lower bound after 5 rounds. Then, we focus on this class and evaluate the lower bound of the number of active S-boxes after 6 rounds, which will be repeated until the 8th round.

As a result, we find three nibble permutations that can achieve 60 active S-boxes over 8 rounds. Table 6.10 shows the comparison of the lower bound of the number of active S-boxes for Midori-128 and our structure. Compared with Midori-128, our nibble permutations guarantee a much larger number of active S-boxes after 6 rounds, about a factor of 1.5.

6.3.3.7 Hybrid Use of Bit and Nibble Permutations. Consequently, we obtain a set of bit and nibble permutations. For Orthros, we pick a bit and nibble permutations from the set, and use the bit permutation for the first 4 rounds, in order to achieve a fast full diffusion. Specifically, it achieves full diffusion in 2.5 rounds, while Midori-128 requires 3 rounds. For the rest of 8 rounds we used the nibble permutation to guarantee a large number of active S-boxes. Indeed, 10 rounds of Orthros starting from the 3rd round, i.e., the 3rd to the 12th round, achieve 64 active S-boxes. We note that Midori-128 needs 13 rounds to obtain 64 active S-boxes, thus the gain is 3 rounds.

6.3.4 S-box

We search a small-delay and lightweight 4-bit S-box which fulfills the following requirements: (1) the maximal probability of a differential is 2^{-2}, (2) the maximal absolute bias of a linear approximation is 2^{-2} and (3) full diffusion, i.e., any input bit difference diffuses to all output bits. We use a metric called *depth* [11] to estimate the path delay of S-boxes.

Definition 16 (depth): *The depth is defined as the sum of the sequential path delays of basic operations, namely* `AND`, `OR`, `NAND`, `NOR` *and* `NOT`.

Following the assumption of [11], our search assumes that depths of `XOR`, `AND/OR`, `NAND/NOR`, and `NOT` are weighted as 2, 1.5, 1 and 0.5, respectively, and the required gates of `NOT`, `NAND/NOR`, `AND/OR` and `XOR/XNOR` are estimated as 0.5, 1, 1.5 and 2 Gate Equivalents (GEs), respectively. We search over the set of all 4-bit S-boxes, whose size is $2^{44.3}$, sort them in order of small depth, and check whether they satisfy our security requirements.

We remark that our construction does not require the involution property of S-box unlike Midori's Sb_1. It allows us to expand the number of possible candidates from $2^{25.5}$ (the number of all involution 4-bit S-boxes) to $2^{44.3}$. As a result, we found an S-box (see Table 6.1) whose depth and gate size are the lowest and the smallest in our search. Specifically, the depth is 3.5 and the area is 20 GE under the aforementioned assumption of [11]. The S-box can be expressed as

Table 6.11: Comparison of S-boxes.

	Orthros	Midori [11]	QARMA [9]				PRINCE [53]	
	S	Sb_1	σ_0	σ_1	σ_2	σ_2^{-1}	S	S^{-1}
Area Optimized								
Area [GE]	**10.7**	12.0	11.0	12.2	15.7	15.5	12.2	15.5
Delay [ps]	**285.7**	367.2	380.6	412.3	328.3	531.3	341.9	390.7
Delay Optimized								
Area [GE]	**40.4**	32.2	43.2	36.4	40.2	65.4	51.1	41.2
Delay [ps]	**37.2**	71.1	39.2	89.3	58.6	53.3	52.7	80.6
Full diffusion	**Yes**	Yes	No	Yes	Yes	Yes	Yes	Yes
Involution	No	Yes	Yes	Yes	No	No	No	No

follows, where inputs and outputs are defined as $\{x_0, x_1, x_2, x_3\}$ and $\{y_0, y_1, y_2, y_3\}$, and x_3 and y_3 are the most significant bits.

$$y_0 = \Big((x_0 \texttt{ NOR } x_3) \texttt{ AND } x_1\Big) \texttt{ NOR } \Big((x_1 \texttt{ NAND } x_2) \texttt{ AND } x_0\Big)$$
$$y_1 = \Big((x_1 \texttt{ NOR } x_2) \texttt{ OR } x_0\Big) \texttt{ NAND } \Big((x_0 \texttt{ NAND } x_3) \texttt{ OR } x_2\Big)$$
$$y_2 = \Big((x_1 \texttt{ OR } x_3) \texttt{ NAND } x_2\Big) \texttt{ NAND } (x_0 \texttt{ NAND } x_1)$$
$$y_3 = \Big((x_1 \texttt{ NAND } x_2) \texttt{ NAND } x_3\Big) \texttt{ NAND } \Big((x_0 \texttt{ NAND } x_2) \texttt{ OR } x_3\Big)$$

Compared to the S-box of Midori-128, the depth and area can be reduced to 3.5 and 10.7 GE from 4 and 12 GE, respectively, when synthesized with the standard cell library of the STM 90nm CMOS logic process (as shown in Table 6.11) with area optimization. The table also shows detailed comparisons with S-boxes of the QARMA and PRINCE family when the circuit is optimized with respect to area as well as delay. The S-box of Orthros performs well when optimized across both metrics. Note that σ_0 does not have the full diffusion property.

6.3.5 Key Scheduling Function

To minimize the hardware cost, key scheduling functions of Orthros are realized by only bit permutations, whose hardware overhead, such as area and delay, is essentially free. We use a class of bit permutations that satisfy both Condition 1 and 2 in the key scheduling functions of Branch1 and Branch2 as shown in Table 6.3, although it cannot guarantee the full diffusion property as there is no S-box and Matrix in the key scheduling functions. One reason to introduce two different key scheduling functions for each branch is to increase the hardness of the key-recovery attack, as will be discussed in the next section.

6.4 Security Evaluation

6.4.1 Differential/Linear Attack

To evaluate the resistance against differential attacks and linear attacks, one way is to obtain the lower bound of the number of differentially and linearly active S-boxes in each round, which can be efficiently computed with a MILP-based method [133]. In the following, we will present lower bounds of the number of differentially and linearly active S-boxes for Branch1, Branch2 and the whole Orthros. Since the maximal differential and linear probability of the S-box is 2^{-2}, it is sufficient to guarantee the security against differential attacks and linear attacks if there are 64

active S-boxes, as it gives $2^{-2\times 64} = 2^{-128}$ as an estimate of a differential probability. In our evaluation, we only consider the single-key setting.

As discussed in Section 6.3.3, we observed that it is computationally infeasible to obtain a lower bound for more than 5 rounds starting from the first round of Branch1, Branch2 and Orthros, even with a computer equipped with 48 cores and 256 GB RAM. The search space of 128-bit bitwise differential and linear trails for the first 4 rounds is huge. On the other hand, for the last 8 rounds of Branch1 and Branch2 starting from the 5th round, where the nibble permutation is adopted, we can obtain tight lower bounds of the number of active S-boxes for the nibble-wise differential and linear trails. In addition, we can obtain tight lower bounds of the number of active S-boxes of 5 rounds of Orthros starting from the 5th round, i.e., 5 to 9 rounds.

In our evaluation, each of Branch1, Branch2 and Orthros is first divided into two parts, i.e., the first 4 rounds and the remaining 8 rounds. We compute a lower bound of the number of active S-boxes for each part. The lower bound of Orthros is obtained by the sum of those of Branch1 and Branch2.

The corresponding results are displayed in Table 6.12. Table 6.12 shows that there are at least 68 active S-boxes in 5 rounds of Orthros starting from the 5th round, i.e., 5 to 9 rounds. In addition, the last 10 rounds of Branch1 and Branch2 including 2 bit-permutation rounds and 8 nibble-permutation rounds, i.e., 3 to 12 rounds, have at least $64(= 4 + 60)$ active S-boxes. Although we do not claim any security for Branch1 and Branch2 as a full-fledged block cipher, each has a sufficient number of active S-boxes in the full 12 rounds.

Table 6.12: The lower bounds of the number of active S-boxes in the single-key setting.

Construction	**bit/nibble**	**Rounds**											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Orthros (the first 4 rounds)	bit	2	8	12	16	-	-	-	-	-	-	-	-
Orthros (from the 5th round)	nibble	-	-	-	-	2	8	20	40	**68**	72	101	120
Branch1 (the first 4 rounds)	bit	1	4	6	8	-	-	-	-	-	-	-	-
Branch2 (the first 4 rounds)	bit	1	4	5	8	-	-	-	-	-	-	-	-
Branch1 (from the 5th round)	nibble	-	-	-	-	1	4	7	16	25	36	50	60
Branch2 (from the 5th round)	nibble	-	-	-	-	1	4	7	16	25	36	51	60

It should be emphasized that the lower bounds of Orthros in Table 6.12 are not tight i.e., actually the full rounds of Orthros includes more active S-boxes. This is because the number of active S-boxes of Orthros after 10 rounds is computed as the sum of those of Branch1 and Branch2. Besides, those of first 4 rounds and the last 8 rounds are independently obtained. Thus, we expect that the full-round Orthros can resist against the differential attack and the linear attack.

6.4.2 Impossible Differential Attack

The impossible differential attack can be estimated by the required number of rounds for the full diffusions. In the forward direction, both Branch1 and Branch2 require 2.5 rounds for the full diffusion, while it is 5 rounds in the backward direction. Consequently, we expect that there is no any probability-one impossible differential characteristic over 8 rounds of Branch1 and Branch2, respectively. Since Orthros take a sum of the outputs of Branch1 and Branch2, we believe that the number of rounds of an impossible differential characteristic for Orthros is much lower than that of Branch1 and Branch2.

To obtain actual impossible differential characteristics, we utilize the MILP-aided automatic searching tool proposed by Sasaki and Todo [158]. Taking DDT (differential distribution table) of the S-box into consideration, we searched bit-wise impossible differential characteristics of Orthros that have one active bit for both of a plaintext and a ciphertext. The details of DDT and

modeling S-box are as follows:

DDT and modeling of S-box The aforementioned DDT table of our S-box is shown in Table 6.13, where d_{in} and d_{out} denote the input and output difference of 4-bit S-box, respectively.

Table 6.13: The differential distribution table of S-box.

d_{in} \ d_{out}	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1	0	2	2	4	0	2	2	0	0	0	0	4	0	0	0	0
0x2	0	0	2	2	4	4	0	0	2	2	0	0	0	0	0	0
0x3	0	2	2	0	0	2	0	2	0	0	2	2	0	0	4	0
0x4	0	0	2	0	2	0	4	0	2	2	0	2	2	0	0	0
0x5	0	0	2	2	0	4	0	0	0	4	0	0	0	0	2	2
0x6	0	2	2	0	2	0	0	2	0	0	0	0	2	2	2	2
0x7	0	2	0	0	0	0	2	4	0	0	2	0	4	2	0	0
0x8	0	2	0	0	2	0	0	0	4	2	4	0	2	0	0	0
0x9	0	0	0	2	0	0	0	2	0	2	2	2	2	0	0	4
0xa	0	2	0	0	2	0	0	0	0	2	2	2	0	2	4	0
0xb	0	0	2	0	0	0	2	0	2	0	2	2	0	2	0	4
0xc	0	2	2	2	0	2	0	0	0	0	2	0	2	2	2	0
0xd	0	2	0	2	0	0	2	2	2	2	0	0	0	2	0	2
0xe	0	0	0	0	0	2	4	2	4	0	0	0	0	2	0	2
0xf	0	0	0	2	4	0	0	2	0	0	0	2	2	2	2	0

Based on the method of [166], we derive following 37 linear inequalities from Table 6.13, where $d_{in} = (d_{in_0}, d_{in_1}, d_{in_2}, d_{in_3})$ and $d_{out} = (d_{out_0}, d_{out_1}, d_{out_2}, d_{out_3})$.

$$
\begin{cases}
-d_{in_0} - d_{in_1} + 2d_{in_2} - d_{in_3} - 2d_{out_0} + 2d_{out_1} - d_{out_2} + d_{out_3} + 3 & \geq 0 \\
d_{in_0} + d_{in_1} - d_{in_2} + d_{in_3} + d_{out_0} + d_{out_1} + d_{out_2} & \geq 0 \\
d_{in_0} + 2d_{in_2} + d_{in_3} + d_{out_0} - d_{out_1} - d_{out_2} - 2d_{out_3} + 2 & \geq 0 \\
2d_{in_0} - d_{in_1} + d_{in_2} + 2d_{in_3} + d_{out_0} - d_{out_1} + d_{out_2} - 2d_{out_3} + 2 & \geq 0 \\
d_{in_0} - 2d_{in_1} - 2d_{in_2} - d_{in_3} + 2d_{out_0} - 2d_{out_1} + 2d_{out_2} - d_{out_3} + 6 & \geq 0 \\
-d_{in_0} + 2d_{in_1} - 2d_{in_2} - d_{in_3} + 2d_{out_0} + d_{out_1} - d_{out_2} - 2d_{out_3} + 5 & \geq 0 \\
d_{in_0} + d_{in_1} + 2d_{in_2} + 2d_{in_3} - d_{out_1} - d_{out_3} & \geq 0 \\
d_{in_0} + 2d_{in_1} + d_{in_2} + 2d_{in_3} - d_{out_0} - d_{out_1} - d_{out_2} - d_{out_3} + 1 & \geq 0 \\
d_{in_0} + d_{in_1} - d_{in_2} + d_{out_0} - d_{out_1} - d_{out_2} + d_{out_3} + 2 & \geq 0 \\
d_{in_1} + d_{in_2} + 2d_{in_3} - d_{out_1} - d_{out_2} - d_{out_3} + 1 & \geq 0 \\
2d_{in_0} + d_{in_2} + d_{in_3} - 2d_{out_0} + d_{out_1} - 2d_{out_2} + 2d_{out_3} + 2 & \geq 0 \\
d_{in_0} - d_{in_2} - d_{in_3} - d_{out_0} - d_{out_1} - d_{out_2} - d_{out_3} + 5 & \geq 0 \\
d_{in_0} + 2d_{in_1} + 2d_{in_2} - 2d_{out_0} - 1d_{out_1} - 1d_{out_2} + d_{out_3} + 2 & \geq 0 \\
-d_{in_0} + 2d_{in_1} + d_{in_2} + d_{in_3} + 2d_{out_0} - 2d_{out_2} + d_{out_3} + 1 & \geq 0 \\
d_{in_0} + d_{in_1} + d_{in_2} - d_{in_3} - d_{out_0} + d_{out_1} + d_{out_2} + d_{out_3} & \geq 0 \\
d_{in_0} + d_{in_1} + d_{in_2} - 2d_{out_0} - d_{out_1} + d_{out_2} - d_{out_3} + 2 & \geq 0 \\
-d_{in_0} + d_{in_1} + d_{in_3} + d_{out_0} - d_{out_1} - d_{out_3} + 2 & \geq 0 \\
-d_{in_0} - d_{in_1} + d_{in_2} - d_{out_0} + d_{out_1} - d_{out_2} - d_{out_3} + 4 & \geq 0 \\
d_{in_0} - d_{in_2} + d_{in_3} + d_{out_0} - d_{out_1} - d_{out_2} + d_{out_3} + 2 & \geq 0 \\
d_{in_0} - d_{in_1} - d_{in_2} + d_{out_1} - d_{out_2} - d_{out_3} + 3 & \geq 0 \\
-d_{in_1} - 2d_{in_2} - d_{in_3} - 2d_{out_0} + 2d_{out_1} + 2d_{out_2} - d_{out_3} + 5 & \geq 0 \\
-3d_{in_0} - 2d_{in_1} - 3d_{in_2} + d_{in_3} + 4d_{out_0} + 3d_{out_1} + d_{out_2} + 2d_{out_3} + 4 & \geq 0 \\
2d_{in_0} - d_{in_1} + d_{in_2} - 3d_{in_3} + 2d_{out_0} + d_{out_1} + 4d_{out_2} + 3d_{out_3} & \geq 0 \\
d_{in_0} - 2d_{in_1} - 2d_{in_2} - 2d_{out_0} + 2d_{out_1} + d_{out_2} - 1d_{out_3} + 5 & \geq 0 \\
-3d_{in_0} + d_{in_1} + d_{in_2} - 2d_{in_3} + 2d_{out_0} - 2d_{out_1} + 3d_{out_2} - 1d_{out_3} + 5 & \geq 0 \\
-2d_{in_0} - 2d_{in_1} + d_{in_2} + d_{in_3} + 3d_{out_0} + 2d_{out_1} + 4d_{out_2} + 4d_{out_3} & \geq 0 \\
d_{in_0} - d_{in_1} + 2d_{in_2} - 3d_{in_3} + d_{out_0} - d_{out_1} + 2d_{out_2} + 3d_{out_3} + 2 & \geq 0 \\
-d_{in_0} + d_{in_1} - 2d_{in_2} + d_{in_3} - 2d_{out_0} - d_{out_1} + 2d_{out_2} + 2d_{out_3} + 4 & \geq 0 \\
-2d_{in_0} - 2d_{in_1} + d_{in_2} + d_{in_3} + 2d_{out_0} - 1d_{out_1} + d_{out_2} + 2d_{out_3} + 3 & \geq 0 \\
-d_{in_0} - 2d_{in_1} - d_{in_2} + d_{in_3} - d_{out_0} + 2d_{out_1} + d_{out_2} - 2d_{out_3} + 5 & \geq 0 \\
d_{in_1} - d_{in_2} - d_{out_0} - d_{out_1} + d_{out_2} + d_{out_3} + 2 & \geq 0 \\
-d_{in_0} - d_{in_1} - d_{in_2} + d_{in_3} - d_{out_0} - d_{out_2} + d_{out_3} + 4 & \geq 0 \\
-3d_{in_0} - 4d_{in_1} - 2d_{in_2} - 2d_{in_3} + d_{out_0} + 3d_{out_1} - d_{out_2} + 4d_{out_3} + 8 & \geq 0 \\
-d_{in_1} - d_{in_2} - d_{in_3} - d_{out_0} - d_{out_1} - d_{out_2} - d_{out_3} + 6 & \geq 0 \\
-d_{in_0} + d_{in_1} - d_{in_3} - d_{out_0} - d_{out_1} - d_{out_2} + d_{out_3} + 4 & \geq 0 \\
d_{in_0} - d_{in_1} - d_{in_2} - d_{in_3} - d_{out_0} - d_{out_1} - d_{out_2} + 5 & \geq 0 \\
-d_{in_0} + d_{in_2} - d_{in_3} - d_{out_0} - d_{out_1} - d_{out_2} + d_{out_3} + 4 & \geq 0
\end{cases}
$$

As a result, we found 3/5/5-round impossible differential characteristics of Orthros, Branch1

and Branch2, respectively, as follows:

$$\text{Orthros:}\quad \overset{\text{Input}}{\begin{bmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\end{bmatrix}} \xrightarrow{3\text{ rounds}} \overset{\text{Output}}{\begin{bmatrix} 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\end{bmatrix}},$$

$$\text{Branch1 :}\quad \overset{\text{Input}}{\begin{bmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\end{bmatrix}} \xrightarrow{5\text{ rounds}} \overset{\text{Output}}{\begin{bmatrix} 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\end{bmatrix}},$$

$$\text{Branch2 :}\quad \overset{\text{Input}}{\begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\end{bmatrix}} \xrightarrow{5\text{ rounds}} \overset{\text{Output}}{\begin{bmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\end{bmatrix}}.$$

Thus, we expect that the full-round Orthros is secure against impossible differential attacks.

6.4.3 Integral Attack

We present integral distinguishers on round-reduced Orthros. Since the division property [170, 171] was proposed, it has become an efficient tool to evaluate the resistance against integral attacks. Moreover, with the development of the MILP model for the bit-based division property [177], the attacker now only needs to focus on modeling the propagation of the division property.

The round function of Orthros consists of a nonlinear layer (`S-box`), a bit/nibble permutation layer, another linear layer (`matrixMul`) and the constant/key addition. To model the propagation of the division property through each component, we only need to consider the S-box and the binary matrix used in `matrixMul`. The bit/nibble permutation only has an influence on the coordinates of the variables used in the MILP model. The modelling of the S-box and binary matrix are

as follows:

Modeling S-box. Similar to Section 6.4.2, one could build a table to describe the propagation of the division property through `S-box`, as shown in Table 6.14. In this table, u and v denote

Table 6.14: The propagation of the division property for the S-box.

u \ v	0x0	0x1	0x2	0x4	0x8	0x3	0x5	0x9	0x6	0xa	0xc	0x7	0xb	0xd	0xe	0xf
0x0	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x1		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x2		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x4		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x8		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x3		*	*	*		*	*	*	*	*	*	*	*	*	*	*
0x5		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x9		*	*		*	*	*	*	*	*	*	*	*	*	*	*
0x6		*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
0xa		*		*	*	*	*	*	*	*	*	*	*	*	*	*
0xc			*	*	*	*	*	*	*	*	*	*	*	*	*	*
0x7		*	*	*		*	*	*	*	*	*	*	*	*	*	*
0xb		*				*	*	*	*	*	*	*	*	*	*	*
0xd			*		*	*	*	*	*	*	*	*	*	*	*	*
0xe				*	*		*	*	*	*	*	*	*	*	*	*
0xf																*

the input and output division property of `S-box`, respectively. The entry at (u, v) is $*$ when the propagation $u \to v$ is possible. Otherwise, the propagation is impossible. Based on the method proposed by [166], such a table is equivalent to the linear inequalities as shown below, where $u = (u_0, u_1, u_2, u_3)$ and $v = (v_0, v_1, v_2, v_3)$.

$$\begin{cases} -u_0 - u_1 + 2v_0 + 2v_1 + 2v_2 + v_3 & \geq 0 \\ -u_0 - u_2 + v_0 + v_1 + v_3 + 1 & \geq 0 \\ -u_2 - u_3 + v_0 + 2v_1 + 2v_2 + 2v_3 & \geq 0 \\ -u_0 - u_1 - u_3 + 3v_0 + 2v_1 + 3v_2 + 2v_3 & \geq 0 \\ -u_0 - u_3 + 2v_0 + v_1 + 2v_2 + 2v_3 & \geq 0 \\ -u_0 - u_1 - u_2 + v_0 + v_1 + 2 & \geq 0 \\ -4u_0 - 3u_1 - 4u_2 - 4u_3 + v_0 + v_1 + v_2 + 2v_3 + 10 & \geq 0 \end{cases}$$

Modeling the Binary Matrix. For the `matrixMul` operation, it can also be viewed that the binary matrix M_b works on four bits independently. Let $B = M_b \cdot A$, where $A, B \in \mathbb{F}_2^4$. Therefore, we could pre-compute a table to describe the mapping from A to B, as specified in Table 6.15.

Table 6.15: The mapping of the binary matrix.

A	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
B	0	14	13	3	11	5	6	8	7	9	10	4	12	2	1	15

To model the propagation of the division property through the binary matrix, similar to the way we model `S-box`, we first build a table to describe the propagation rule (Table 6.16), where w and z denote the input and output division property. The entroy (w, z) is marked

as $*$ when the propagation $w \to z$ is possible. Otherwise, the propagation is impossible. By using the method of [166], Table 6.16 can be expressed by the linear inequalities as

Table 6.16: The propagation of the division property for the binary matrix.

w \ z	0x0	0x1	0x2	0x4	0x8	0x3	0x5	0x9	0x6	0xa	0xc	0x7	0xb	0xd	0xe	0xf
0x0	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
0x1			$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
0x2		$*$		$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
0x4		$*$	$*$		$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
0x8		$*$	$*$	$*$		$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
0x3						$*$	$*$	$*$	$*$	$*$		$*$	$*$	$*$	$*$	$*$
0x5						$*$	$*$	$*$	$*$		$*$	$*$	$*$	$*$	$*$	$*$
0x9						$*$	$*$	$*$		$*$	$*$	$*$	$*$	$*$	$*$	$*$
0x6						$*$	$*$		$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
0xa						$*$		$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
0xc							$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$	$*$
0x7													$*$	$*$	$*$	$*$
0xb												$*$		$*$	$*$	$*$
0xd												$*$	$*$		$*$	$*$
0xe												$*$	$*$	$*$		$*$
0xf																$*$

displayed below, where $w = (w_0, w_1, w_2, w_3)$ and $z = (z_0, z_1, z_2, z_3)$.

$$\begin{cases} -w_0 - w_1 - w_2 + z_3 + 2 & \geq 0 \\ -w_0 - w_1 - w_3 + z_2 + 2 & \geq 0 \\ -w_0 - w_1 - w_2 - w_3 + z_0 + z_1 + z_2 + z_3 & \geq 0 \\ -w_0 - w_2 - w_3 + z_1 + 2 & \geq 0 \\ -w_1 - w_2 - w_3 + z_0 + 2 & \geq 0 \\ -w_0 - w_2 + z_0 + z_2 + 1 & \geq 0 \\ -w_0 - w_1 + z_0 + z_1 + 1 & \geq 0 \\ -w_1 - w_2 + z_1 + z_2 + 1 & \geq 0 \\ -w_1 - w_3 + z_1 + z_3 + 1 & \geq 0 \\ -w_2 + z_0 + z_1 + z_3 & \geq 0 \\ -w_3 + z_0 + z_1 + z_2 & \geq 0 \\ -w_2 - w_3 + z_2 + z_3 + 1 & \geq 0 \\ -w_1 + z_0 + z_2 + z_3 & \geq 0 \\ -w_0 + z_1 + z_2 + z_3 & \geq 0 \\ -w_0 - w_3 + z_0 + z_3 + 1 & \geq 0 \end{cases}$$

Based on our model, the longest integral distinguisher can reach up to at most 7 rounds with 127 active bits in the input. For example, when the most significant bit of the plaintext is constant and the remaining 127 bits take all possible 2^{127} values, for 7-round Orthros, all output bits are balanced.

6.4.3.1 Remark. Although there are several 7-round integral distinguishers, it is difficult to mount a key-recovery attack on 8 rounds of Orthros. This is different from usual key-recovery

attacks on block ciphers, where the attacker is able to add several rounds after the integral distinguisher and guess partial key bits to decrypt the ciphertext. It is quite costly to guess the key bits and reverse the ciphertext for Orthros since the final output is the sum of the outputs of two branches, i.e., the attacker further needs to guess the output of the other branch.

6.4.4 Invariant Subspace Attack

Beierle et al. [25] showed that an invariant subspace attack can be mounted on a block cipher if one finds a non-trivial invariant for the substitution layer whose linear space is invariant under the linear layer matrix L that it uses and contains all the differences between the round keys. For block ciphers without a dedicated key schedule function, say when the i-th round key $rk_i = k \oplus rc_i$ is simply the xor of the master key and the i-th round constant, the difference of all round keys is the difference of the round constants. If D denotes the set of all round constant differences, the authors of [25] computed $W_L(D)$, which denotes the smallest L-invariant subspace containing D. If the dimension of $W_L(D)$, $\dim(W_L(D))$, satisfies $\dim(W_L(D)) \geq n-1$, where n is the block size of the cipher in bits, then the authors showed that there is no non-trivial invariant of the substitution layer, provided that the S-box is well designed and does not have any linear component.

If this condition is not satisfied, one must further investigate the properties of the substitution layer. The authors then showed that, for every subspace Z of the 0-linear space of the invariant of the substitution layer S, the invariant, g, takes the same value on each coset of Z in $\mathbb{F}_2^n$ and also on each element of the set $S(Z)$. To show that g is trivial, the authors computed the S-box layer at some points in Z and hoped that all cosets would be hit when evaluating S at a few points in Z. If g takes the same value on all the corresponding cosets, we would conclude that g must be a constant function and thus trivial. This can be done if we take $Z = W_L(D)$ and if $\dim(W_L(D))$ is close to n, since one would only need to hit $2^{n-\dim(W_L(D))}$ cosets.

Since our construction uses a key schedule function for both branches, we can not directly construct the set D as the difference of round constants. From this fact, we use four different linear layers (two different in each branch) in our construction. However for any randomly chosen value of the secret key k one can construct the sets D_1, D_2, D_3, D_4, one each for the difference of the round keys used in each of the four linear layers L_1, L_2, L_3, L_4, and then try to compute $\dim(W_{L_i}(D_i))$ for each i. We found that the linear matrices composed by a bit permutation and a matrix multiplication (used in the first to 4th rounds of each branch, call them L_1 (left), L_3 (right)) have extremely high multiplicative orders – around 2^{48} to 2^{60} – and thus it is not directly possible to find $W_L(D)$ for these matrices. Thus we limited ourselves to find $W_L(D)' = \sum_{c \in D} < L^i(c),\ i \leq 10000 >$ for L_1, L_2 and L_3 (where $< \cdot >$ denotes the subspace generated by the constituent vectors). We did an experiment with 1000 randomly chosen keys, and computed $W_{L_i}(D_i), W_{L_i}(D_i)'$. The dimension of these spaces is almost always more than 127 for L_1, L_3 and always more than 123 for L_2, L_4. Whenever the dimension of these spaces was less than 127, we tried to run Algorithm 1 of [25] to see if all cosets are hit when trying to evaluate the S-layer. For all choices of the random key, we find that all the cosets are always hit, and thus we conclude that it is highly unlikely that an invariant subspace attack can be mounted on our construction.

6.4.5 Meet-in-the-Middle Attack

To mount a meet-in-the middle attack, the adversary has to compute the inverse of the cipher, i.e., computing intermediate states (matching states) from the corresponding ciphertexts by guessing the involved round keys. As discussed above, the adversary needs to guess one of two 128-bit outputs of branches to go through the final XOR operation in the backward computation. As it requires at least 2^{128} iterations, the attack is not efficient than the brute force attack.

Another possibility is to use the splice-and-cut technique [157]. With this technique, the adversary guesses the intermediate values as the start point from which the adversary starts the computation toward both directions. However, once she guesses a 128-bit intermediate of one branch, she has to correctly guess the corresponding 128-bit intermediate of the other branch. Therefore, we believe that Orthros is secure against meet-in-the middle attacks.

6.4.6 Yoyo and Mixture-Differential Attacks

The yoyo attack was first introduced by Biham [40]. Recently, it has been applied to the cryptanalysis of AES by Rønjom et al. [151], where generic attacks on up to 3 rounds of SPNs have been discussed. Since 2-round AES can be viewed as 1-round SPN with the concept of super S-box, distinguishing attack on up to 6 rounds of AES are derived by [151]. However, there is one major step in the yoyo attack, that is, the attacker needs to make a decryption query. For the design of Orthros, since the final output is the xor sum of the outputs of Branch1 and Branch2, it is infeasible to make a decryption query. In addition, due to the fast diffusion of the bit permutation and the fact that each branch adopts a different bit permutation, it is quite difficult to construct an efficient super S-box for Orthros in the first four rounds. Based on these reasons, we believe that Orthros is resistant against such an attack.

In Asiacrypt 2019, a different view of the yoyo attack on AES, called exchange attack, was proposed by Bardeh and Rønjom [23]. It does not require decryption queries and can reach up to 6 rounds of AES. However, due to the similarity in the underlying idea between the yoyo attack and exchange attack, we believe that the resistance against the yoyo attack implies the resistance against exchange attack.

The mixture differential introduced by Grassi [85] is an efficient tool to analyze a reduced-round AES, as its contribution to the recent progress of key-recovery attacks on 5-round AES [22, 77]. An important factor which makes the mixture differential efficient is that AES adopts a word-wise permutation. Due to the effect of the bit permutation in the first four rounds, we are not able to find a useful mixture differential for Orthros.

6.4.7 Difficulty of Key-Recovery Attacks

As we repeated several times, the unique feature of Orthros (as a cryptographic primitive) is that it takes the sum of two branch outputs. To mount a key-recovery attack with a statistical distinguisher for block ciphers, such as a differential/linear/integral distinguisher, it is common to append a few rounds after a certain number of rounds for which a distinguisher exists, and guess partial key bits by partially decrypting the ciphertext and verifying whether the distinguishing property holds. However, such a common strategy is quite hard for Orthros since the attacker even needs to guess the outputs of each branch in order to reverse the ciphertext. In addition, it is required to construct two distinguishers for two different branches with the same plaintext set simultaneously if the attacker wants to append a few rounds after the distinguishable rounds. Even if it is possible to construct a distinguisher for one block cipher with an advanced cryptanalysis method as discussed above, it would be challenging to construct two different distinguishers for two different block ciphers for the same plaintext set simultaneously. In such a situation, we think generally the most promising direction is to find integral distinguishers. This will be discussed later.

Another attacking strategy is to prepend some rounds before the distinguishable rounds. However, this implies that there exists a distinguisher for each branch simultaneously. When extending two distinguishers backwards to the plaintext, the attacker can derive which key bits should be guessed in order to compute the desired value of the intermediate internal state of both branches. Since each branch adopts a different linear layer in its round function, and the key schedules of two branches also differ, a lot of to-be-guessed key bits will be involved. Moreover, the whiten-

ing keys in two branches are different as well, which further increases the complexity to prepend some rounds before a distinguisher. For better understanding, we present a framework to recover the secret key by extending an integral distinguisher backwards.

6.4.7.1 A Framework for Recovering the Secret Key. This framework was once applied to a preliminary version of Orthros. Therefore, we omit the details of the design and explain a high-level idea. First, we denote the states after the S-box layer in the first round of Branch1 and Branch2 by $XL^{0.5}$ and $XR^{0.5}$, respectively. Suppose there is a set of bit positions denoted by $PSet \subseteq \{i \mid 0 \leq i \leq 127\}$ and the size of $PSet$ is $PSize$. In addition, let us denote the final output after r rounds of (an old version of) Orthros by C^r, which is the sum of the outputs of (old versions of) Branch1 and Branch2. Suppose there is an integral distinguisher such that

$$\sum C^r = \sum_{XL^{0.5} \in PS} \mathsf{Branch1}(XL^{0.5}) \oplus \sum_{XR^{0.5} \in PS'} \mathsf{Branch2}(XR^{0.5}) = 0,$$

where $PS, PS' \in \{e \in \mathbb{F}_2^{128} \mid e[i] \in \{0,1\}, i \in PSet\}$, i.e., the set of values whose bits located at the positions belonging to $PSet$ take all the possible 2^{PSize} values and the remaining $(128 - PSize)$ bits take constant values.

To mount a key recovery attack, the attacker first derives from $PSet$ the active bits in the plaintext. Specifically, if $i \in PSet$, the $(4 \times i/4)$-th, $(4 \times i/4 + 1)$-th, $(4 \times i/4 + 2)$-th and $(4 \times i/4 + 3)$-th bits of the plaintext are all active. Let $ASize$ be the size of the active bits in the plaintext. The attacker then prepares a plaintext set whose active bits take all possible values and make encryption queries with the r-round Orthros. It is easily detected that the sum of the ciphertext is zero. Record the corresponding 2^{ASize} pairs of plaintext and ciphertext in a table.

Suppose the whitening keys used by Branch1 and Branch2 are the same. In this case, the attacker guesses 2^{ASize} different values of the whitening key which is xored with the active bits in the input. For each guess, the attacker can partially know the corresponding $XL^{0.5}$ and $XR^{0.5}$ and can divide the plaintext set into $2^{ASize-PSize}$ different subsets according to the value of the nonactive bits of $XL^{0.5}$ and $XR^{0.5}$ via the constructed integral distinguisher. For each subset, compute the sum of the corresponding ciphertexts. For the correct key, the sum will be zero for all subsets. However, for a wrong key, the sum is zero for a subset with a probability 2^{-128}. Therefore, the attacker can recover the key bits by checking the sum of the ciphertexts for the plaintexts in each subset.

Consider the case when different whitening keys are used for Branch1 and Branch2. In this case, when the attacker guesses the key bits in the left branch to obtain the corresponding $2^{ASize-PSize}$ subsets of the plaintexts, the sum of the ciphetexts for the plaintexts in each subset is not clear even the guess is correct. This is because the set of $XR^{0.5}$ behaves randomly for each subset of plaintext obtained according to the guess of the whitening key used in Branch1.

Obviously, it can be interpreted that this framework for Orthros is to convert a r-round distinguisher into a r-round key-recovery attack. Therefore, a long distinguisher should be prevented in our design.

6.5 Hardware Evaluation

Since our target construction is a low-latency PRF, the most useful hardware evaluation of the design is a fully unrolled circuit that optimizes the signal delay from the input to output ports. Such a circuit would be able to evaluate the PRF in one clock cycle itself, and naturally the clock frequency can be increased till the clock period is just above the total critical path of the circuit, affording a maximum throughput of $\frac{blocksize}{critical\ path}$ bits per second. To perform a fair evaluation we compare our construction with two other low-latency primitives that provide at least 128-bit

block and 128-bit security. The first is Midori-128 and the second is QARMA$_9$-128-σ_0 (in [9], QARMA with 9 forward and backward rounds was recommended for applications targeting 192 bit security). QARMA$_9$-128-σ_0 is a particular instantiation of QARMA with 9 forward and backward rounds and a low-delay S-box σ_0. For an added comparison, we also include the 64-bit block cipher PRINCE in our results. On the other hand we also benchmark some permutation based constructions that can be used as a PRF. For example the Kangaroo12-XOF [38] which is based on the 12-round Keccak-f[1600] permutation can be used as a PRF: one could absorb the key and plaintext in the permutation state and extract 128 bits from the resulting XOF. Let us call this construction Kangaroo12-PRF[1600]. Since any design based on a 1600-bit state would naturally be hardware-intensive we also consider a lightweight version of the above construction Kangaroo12-PRF[400] based on the 12-round Keccak-f[400] permutation. We can also use the Subterranean-Deck function [65] to extract a 128 bit MAC from a 128 bit key and message. We also benchmark this design which we call Subterranean-PRF. It is even more lightweight as it has only a 257-bit state.

We found that across all libraries, Orthros even outperforms PRINCE (see Tables 6.17, 6.19, 6.20, 6.18) when it comes to the absolute signal delay between the input/output ports. We remark that PRINCE is a 64-bit block cipher.

For a fair evaluation we adhered to the following design flow for all the ciphers:

1. The RTL source codes for the circuit of the ciphers were first written in the Verilog HDL, and a functional simulation was done using the Modelsim software to ensure correctness.

2. The RTL codes were synthesized by the *Synopsys Design Vision* circuit compiler, with the compiler command set to `compile_ultra`. No other optimizations are done at this stage. For this process we used the standard cell libraries of the following CMOS logic processes: **a)** STM 90nm, **b)** TSMC 90nm, **c)** Nangate 45nm and **d)** Nangate 15nm.

3. A timing simulation was done on the synthesized netlist to confirm the correctness of the design, by comparing the output of the timing simulation with known test vectors.

4. The switching activity of each gate of the circuit was collected while running post-synthesis simulation. The average power was obtained using *Synopsys Power Compiler*, using the back annotated switching activity.

5. Step 2 outputs the critical path of the circuit. We repeat steps 2-4 (for each of the libraries) but this time by asking the circuit compiler to constrain the total signal delay between the input/output ports to some value less than the critical path computed in step 2.

6. We repeat the above processes, with progressively lower values of total signal delay, till such time as the circuit compiler is unable to construct a circuit with given delay. We stop the flow at this point. All results have been tabulated in Tables 6.17, 6.18, 6.19, 6.20

Why area increases with decrease in latency: A cell library typically has several drive strengths of cells that implement a given logic function. These drive strengths correspond to the capacitive load that a cell can drive without excessive delay and with acceptable signal characteristics. Thus when we force the circuit compiler to construct a circuit with lower delay, it starts introducing higher drive strength gates, that typically occupy more area while offering the same functionality. For example, in the TSMC 90nm library 2-input xor gates of drive strength 1, 2 ,4 occupy around 2.5, 3, 5 GE respectively. Thus it is natural for the area of a circuit to progressively increase as we constrain the circuit compiler to construct circuits of increasingly lower delay as shown in Tables 6.17, 6.18, 6.19, 6.20.

Cipher	Area (μm^2)	(GE)	Power (mW)	Energy (pJ)	Latency (ns)	Max TP (Gbps)
Orthros	98150.7	22307	6.647	664.69	10.60	11.246
	99258.2	22559	6.025	602.50	9.00	13.245
	102286.4	23247	6.214	621.40	8.00	14.901
	108582.3	24678	7.220	722.02	7.00	17.030
	123160.6	27991	9.612	961.24	6.00	19.868
	133931.3	30430	10.751	1075.07	5.00	23.842
	148432.8	33735	12.614	1261.35	4.00	29.802
	177991.2	40453	16.591	1659.12	3.00	39.736
	298855.6	67922	27.628	2762.76	2.40	49.671
Midori-128	85435.0	19417	10.205	1020.49	18.54	6.430
	86470.0	19652	9.671	967.14	16.00	7.451
	89648.7	20375	9.748	974.81	14.00	8.515
	96225.5	21869	11.275	1127.54	12.00	9.934
	107393.6	24408	15.687	1568.7	10.00	11.921
	122584.4	27860	17.887	1788.72	8.00	14.901
	144109.4	32752	24.011	2401.09	5.99	19.868
	277950.7	63171	46.464	4646.39	4.10	29.075
QARMA$_9$-128-σ_0	104118.3	23663	10.044	1004.38	19.41	6.142
	104686.9	23792	9.810	981.04	17.00	7.012
	106848.1	24284	9.911	991.05	15.00	7.947
	112157.2	25490	11.571	1157.09	13.00	9.170
	128032.8	29098	16.816	1681.62	11.00	10.837
	147874.2	33608	20.438	2043.83	9.00	13.245
	182268.6	41425	27.714	2771.37	6.96	17.128
	234453.9	53285	41.508	4150.79	4.99	23.890
	319634.3	72644	58.119	5811.87	4.38	27.217
PRINCE	27897.7	6340	1.867	186.73	11.35	10.503
	28773.6	6539	1.791	179.13	9.00	13.245
	31905.0	7251	2.057	205.67	7.00	17.030
	38941.7	8850	3.274	327.44	5.00	23.842
	63795.8	14499	6.023	602.25	3.00	39.736
	87285.5	19838	8.151	815.12	2.56	46.566
Kangaroo12-PRF[1600] *	498909.7	113389	59.533	5953.25	17.52	6.804
	552581.2	125587	58.170	5817.01	12.00	9.934
	590922.6	134301	57.350	5735.03	8.00	14.901
	1184290.6	269157	140.664	14066.38	3.99	29.877
Kangaroo12-PRF[400] *	133662.4	30378	16.674	1667.36	18.47	6.454
	144600.0	32864	14.388	1438.84	12.00	9.934
	167258.9	38013	18.260	1825.96	8.00	28.725
	339374.6	77131	39.311	3931.14	4.15	14.901
Subterranean-PRF *	130206.1	29592	9.976	997.59	17.45	6.831
	139879.2	31791	8.467	846.6	12.00	9.934
	177843.0	40419	18.475	1847.46	8.00	14.901
	375592.1	85362	40.271	4027.13	3.63	32.840

Table 6.17: Results for the STM 90nm library. Power measured at 10 MHz. *The core implementation of the underlying permutations in these constructions were taken from [35, 64]

Cipher	Area (μm^2)	(GE)	Power (mW)	Energy (pJ)	Latency (ps)	Max TP (Gbps)
Orthros	5766.1	29328	2.332	233.18	485.72	245.428
	5792.8	29464	2.187	218.74	450.00	264.910
	6013.5	30586	2.038	203.85	400.00	298.023
	7439.2	37838	2.637	263.73	351.55	339.096
Midori-128	5102.3	25952	3.354	335.36	850.55	140.156
	5116.7	26025	3.251	325.12	800.00	149.012
	5156.9	26229	3.132	313.15	750.00	158.946
	5298.5	26950	3.138	313.80	700.00	170.299
	5731.8	29153	3.305	330.49	650.00	183.399
	6976.4	35484	4.466	446.60	603.78	197.438
QARMA$_9$-128-σ_0	6263.3	31857	4.160	415.98	908.09	131.275
	6304.8	32068	3.810	380.95	800.00	149.012
	7085.8	36040	3.861	386.12	700.00	170.299
	8869.9	45115	5.611	561.05	640.00	186.265
PRINCE	1664.3	8465	0.671	67.14	536.37	222.252
	1671.5	8502	0.657	65.71	500.00	238.419
	1698.5	8639	0.604	60.36	450.00	264.910
	1889.6	9611	0.599	59.98	400.00	298.023
	2337.9	11891	0.894	89.43	371.62	320.783
Kangaroo12-PRF[1600]	26957.8	137114	13.470	1347.04	722.32	165.036
	27298.5	138847	12.223	1222.25	650.00	183.400
	28803.4	146502	12.884	1288.39	600.00	198.682
	31052.8	157943	13.096	1309.60	576.65	206.727
Kangaroo12-PRF[400]	7332.0	37293	3.812	381.22	765.95	155.636
	7407.3	37675	3.711	371.08	700.00	170.300
	7584.7	38578	3.501	350.11	650.00	183.400
	8278.4	42106	5.571	557.07	602.81	197.756
Subterranean-PRF	7582.4	38566	3.548	354.78	692.84	172.059
	7586.4	38586	3.385	338.53	650.00	183.400
	7661.4	38968	3.122	312.22	600.00	198.682
	8392.3	42865	2.878	287.80	543.21	219.453

Table 6.18: Results for the Nangate 15nm library. Power measured at 10 MHz.

Cipher	Area (μm^2)	(GE)	Power (mW)	Energy (pJ)	Latency (ns)	Max TP (Gbps)
Orthros	76712.1	27179	2.452	245.22	9.37	12.722
	77355.6	27407	2.525	252.49	8.00	14.901
	83566.3	29607	3.478	347.79	7.00	17.030
	95896.7	33976	4.447	444.73	6.00	19.868
	124746.6	44197	5.393	539.28	5.29	22.535
Midori-128	67710.8	23990	3.817	381.72	15.89	7.502
	67938.7	24070	4.090	408.97	14.00	8.515
	73455.8	26025	5.374	537.38	12.00	9.934
	90043.0	31902	7.200	719.96	10.00	11.921
	122156.3	43279	9.138	913.82	9.05	13.172
QARMA$_9$-128-σ_0	80258.5	28435	4.2252	422.52	17.40	6.851
	80844.1	28643	5.2578	525.78	15.00	7.947
	89048.8	31550	6.8206	682.06	13.00	9.170
	107814.3	38198	8.4356	843.56	11.00	10.837
	153286.0	54309	10.3168	1031.68	9.43	12.641
PRINCE	22036.6	7807	0.699	69.94	9.79	12.177
	22674.5	8033	0.911	91.09	8.00	14.901
	25879.3	9169	1.135	113.47	7.00	17.030
	31206.6	11056	1.421	142.12	6.00	19.868
	42518.8	15064	1.898	189.81	5.52	21.596
Kangaroo12-PRF[1600]	372818.5	132088	21.558	2155.83	16.21	7.354
	379806.1	134564	20.788	2078.78	14.00	8.515
	424274.4	150319	24.212	2421.24	11.00	10.837
	504947.1	178901	29.376	2937.59	9.12	13.071
Kangaroo12-PRF[400]	104761.8	37117	5.259	525.93	15.15	7.869
	104882.5	37159	5.661	566.09	14.00	8.515
	121185.4	42935	7.248	724.75	11.00	10.837
	151044.3	53514	9.119	911.92	9.27	12.860
Subterranean-PRF	103337.2	36612	3.367	336.72	14.28	8.348
	105067.4	37225	3.388	338.77	12.00	9.934
	123273.3	43675	5.149	514.85	10.00	11.921
	156786.4	55549	5.227	522.74	7.98	14.938

Table 6.19: Results for the TSMC 90nm library. Power measured at 10 MHz.

Cipher	Area (μm^2)	(GE)	Power (mW)	Energy (pJ)	Latency (ns)	Max TP (Gbps)
Orthros	21404.5	26756	15.952	1595.20	3.81	31.289
	21466.2	26833	14.858	1485.80	3.50	34.060
	21548.4	26936	13.386	1338.60	3.00	39.736
	24028.6	30036	13.237	1323.70	2.68	44.481
Midori-128	18784.7	23481	23.088	2308.79	6.29	18.952
	18808.6	23511	22.071	2207.10	6.00	19.868
	18971.7	23715	21.533	2153.30	5.50	21.674
	19666.7	24583	21.014	2101.40	5.00	23.842
	20396.6	25496	21.066	2106.60	4.72	25.256
QARMA$_9$-128-σ_0	22866.4	28583	28.055	2805.50	6.68	17.846
	22903.4	28629	27.896	2789.60	6.50	18.340
	22956.6	28696	25.678	2567.80	6.00	19.868
	23044.9	28806	23.804	2380.4	5.50	21.674
	24699.2	30874	24.174	2417.40	4.92	23.842
PRINCE	6037.4	7547	4.371	437.11	3.87	30.803
	6072.8	7591	4.135	413.54	3.50	34.060
	6473.9	8092	3.994	399.43	3.00	39.736
	6693.1	8366	4.005	400.55	2.92	40.825
Kangaroo12-PRF[1600]	99007.3	124069	85.880	8588.00	5.08	23.466
	101193.1	126808	82.437	8243.70	4.70	25.364
	100167.0	125522	79.505	7950.50	4.50	26.491
	102999.7	129072	78.764	7876.40	4.34	27.468
Kangaroo12-PRF[400]	26726.0	33491	24.496	2449.60	5.49	21.714
	26756.7	33530	23.986	2398.60	5.20	22.925
	27033.3	33876	22.812	2281.20	4.80	24.835
	27699.4	34711	21.814	2181.40	4.50	26.491
Subterranean-PRF	27671.7	34676	22.760	2276.03	5.01	23.794
	27696.2	34707	22.225	2222.50	4.70	25.364
	28071.5	35177	21.406	2140.60	4.30	27.723
	28960.2	36291	20.762	2076.20	3.98	29.952

Table 6.20: Results for the Nangate 45nm library. Power measured at 10 MHz.

For a better illustrative purposes, we provide *Area vs Delay* (see Fig. 6.14). The plots tell us that not only does Orthros perform around 40% better across all standard cell libraries when it comes to the absolute delay value, it also outperforms QARMA$_9$-128-σ_0 and Midori-128 when it comes to achieving **a)** lower area figures and power consumption given a particular delay budget and **b)** lower or competitive delay given a particular area/power budget.

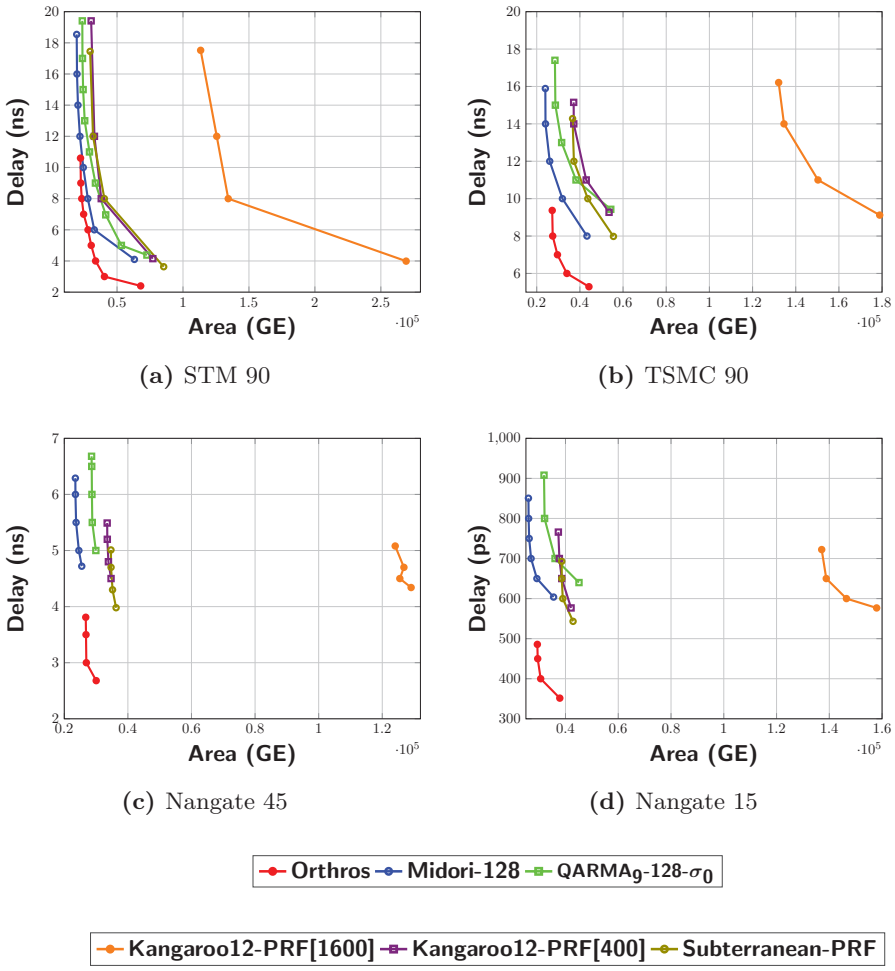


Figure 6.14: Delay vs Area comparisons.

6.6 Conclusions

We have presented a new low-latency PRF of 128-bit block, dubbed Orthros. The design is essentially a sum of keyed permutations, which has been studied in the context of provable security. We found this design is suitable to a low-latency cryptographic primitive, which is intuitive, however, to our knowledge never seriously considered before. We made it real by thoroughly revising the current state-of-the-art low-latency, lightweight building blocks, together with an extensive security analysis and comprehensive hardware benchmarks.

For further directions, it would be interesting to extend our design, say having more branches (with even simpler round functions or fewer rounds), to even reduce latency. Software performance and related-key/side-channel security would also be interesting topics.

7 Rocca: An Efficient AES-Based Encryption Scheme for Beyond 5G

In this section, we present an AES-based authenticated-encryption with associated-data scheme called Rocca, with the purpose to reach the requirements on the speed and security in 6G systems. To achieve ultra-fast software implementations, the basic design strategy is to take full advantage of the AES-NI and SIMD instructions as that of the AEGIS family and Tiaoxin-346. Although Jean and Nikolić have generalized the way to construct efficient round functions using only one round of AES (aesenc) and 128-bit XOR operation and have found several efficient candidates, there still seems to exist potential to further improve it regarding speed and state size. In order to minimize the critical path of one round, we remove the case of applying both aesenc and XOR in a cascade way for one round. By introducing a cost-free block permutation in the round function, we are able to search for candidates in a larger space without sacrificing the performance. Consequently, we obtain more efficient constructions with a smaller state size than candidates by Jean and Nikolić. Based on the newly-discovered round function, we carefully design the corresponding AEAD scheme with 256-bit security by taking several reported attacks on the AEGIS family and Tiaxion-346 into account. Our AEAD scheme can reach 150 Gbps which is almost 5 times faster than the AEAD scheme of SNOW-V. Rocca is also much faster than other efficient schemes with 256-bit key length, e.g. AEGIS-256 and AES-256-GCM. As far as we know, Rocca is the first dedicated cryptographic algorithm targeting 6G systems, i.e., 256-bit key length and the speed of more than 100 Gbps.

7.1 Introduction

7.1.1 Background

The fifth-generation mobile communication systems (5G) have been launched in several countries for commercial services since 2020. Besides, researches for beyond-5G or 6G have been already started in some research institutes. As the first white paper of 6G, [117] was published by the 6Genesisi project in 2019, which is mainly organized by the University of Oulu in Finland. In the white paper, several requirements for 6G systems are raised. For the data transmission speed, it says that 6G achieves more than 100 Gbps, which is more than 10 times faster than that of 5G.

For the 4G system, as underlying cryptographic algorithms to ensure confidentiality and integrity, SNOW 3G [152], AES [135], and ZUC-128 [153] are employed, which are specified as 128-EEA1 (EIA1), 128-EEA2 (EIA2), 128-EEA3 (EIA3), respectively, and these algorithms are also selected cryptographic algorithms for the 5G system as 128-NEA1 (NIA1), 128-NEA2 (NIA2), 128-NEA3 (NIA3). However, for the 5G system, the 3GPP standardization organization requires to increase the security level to 256-bit key lengths. In 2018, ZUC-256 [169] was proposed as the 256-bit key version of ZUC-128. ZUC-256 was revised only in the initialization phase and in the MAC generation phase from ZUC-128. By this revise, ZUC-256 improves the security level against the key-recovery attack to the 256-bit security from the 128-bit security. On the other hand, the performance of the encryption/decryption speed is not quite improved because the keystream generation phase is the same as ZUC-128, and a structural weakness was found [178]. In FSE 2020, Ekdahl *et al.* proposed SNOW-V that is the 256-bit key version of SNOW 3G, and they showed that SNOW-V achieves more than 38 Gbps at an AEAD (Authenticated Encryption with Associated Data) mode on OpenSSL [81]. The performances of SNOW-V are sufficient for them to be used in the 5G system.

However, when taking requirements in 6G systems into account, we have to tackle some challenges. The biggest one is the encryption/decryption speed. For 6G systems, as the data transmission speed is expected to reach more than 100 Gbps, we have to design a cryptographic algorithm

with the encryption/decryption speed of more than 100 Gbps, which is at least three times faster than SNOW-V. Besides, achieving 256-bit security against key-recovery attacks is essential as in 5G systems [3]. In addition, due to the increase of data transmissions in 6G systems, it is necessary to ensure at least 128-bit security against distinguishing attacks while SNOW-V only claims 64-bit security against distinguishing attacks. Therefore, there is no doubt that a new cryptographic algorithm is needed in 6G systems.

For symmetric-key primitives targeting high-performance applications, there are several interesting cryptographic algorithms. The most tempting ones are those employing AES-NI [61, 87], which is a new AES instruction set equipped on many modern CPUs from Intel and AMD. Some SoCs for mobile devices are also equipped with an instruction set for AES [7], and more and more SoCs will support the instruction by the time 6G system is realized. Hence employing AES-NI seems reasonable in designing cryptographic algorithms for 6G systems. The AEGIS family and Tiaoxin-346 belongs to such a category, which are two submissions to the CAESAR competition [2] and AEGIS-128 has been selected in the final portfolio for high-performance applications. The round functions of the AEGIS family and Tiaoxin-346 are quite similar. Specifically, they are only based on the usage of one AES round and the 128-bit XOR operation, both of which have been realized with one instruction on SIMD (Single Instruction, Multiple Data) instructions. As a result, both the AEGIS family and Tiaoxin-346 are competitive in terms of encryption/decryption speed in a pure software environment, if compared with many primitives.

Jean and Nikolić generalized the method to design efficient round functions as that used in AEGIS and Tiaoxin-346 in [103]. After a thorough search, they discovered round functions that can achieve a faster speed than any of the round functions adopted in the AEGIS family and Tiaoxin-346 and provide the 128-bit security against forgery attacks. However, they did not propose a concrete AEAD scheme [103].

Obviously, AEGIS-128, AEGIS-128L and Tiaoxin-346 do not meet the security requirement of the 256-bit key length in 6G systems. In addition, according to our experiments, AEGIS-256 does not reach more than 100 Gbps (See Sect. 7.5). However, those researches leave us the potential of designing the faster cryptographic algorithm based on AES round functions for 6G.

7.1.2 Our Design

In this section, we present an AES-based encryption scheme with a 256-bit key and 128-bit tag called Rocca, which provides both a raw encryption scheme and an AEAD scheme with a 128-bit tag. The goal of Rocca is to meet the requirement in 6G systems in terms of both performance and security. For performance, Rocca achieves an encryption/decryption speed of more than 100 Gbps in both raw encryption scheme and AEAD scheme. For security, Rocca can provide 256-bit and 128-bit security against key-recovery attacks and forgery attacks, respectively.

7.1.2.1 Optimized AES-NI-Friendly Round Function To achieve such a dramatically fast encryption/decryption speed, Rocca is designed for a pure software environment that can fully support both the AES-NI and SIMD instructions. The design of the round function of Rocca is inspired by the work of Jean and Nikolić [103]. To further increase its speed and reduce the state size, we explore a new class of AES-based structures. Specifically, we take the following different approaches.

- To minimize the critical path of the round function, we focus on the structure where each 128-bit block of the internal state is updated by either one AES round or XOR while Jean and Nikolić consider the case of applying both aesenc and XOR in a cascade way for one round, and most efficient structures in [103] are included in this class.

- We introduce a permutation between the 128-bit state words of the internal state in order to increase the number of possible candidates while keeping efficiency as executing such a permutation is a cost-free operation in the target software, which was not taken into account in [103].

We search for round functions that can ensure 128-bit security against forgery attacks in a class of our general constructions as with [103]. Consequently, we succeed in discovering more efficient constructions with a smaller state size than those in [103]. The internal state of Rocca consists of eight 128-bit words and its round function is composed of 4 aesencs and 4 128-bit XOR operations, which is significantly faster than those of the AEGIS family, Tiaxion-346 and Jean and Nikolić's structure [103].

7.1.2.2 Encryption and Authentication Scheme. To resist against the statistical attack in [129], generating each 128-bit ciphertext block will additionally require one AES round, while it is generated with simple quadratic boolean functions in the AEGIS family and Tiaxion-346. However, such a way will have few overhead by AES-NI (See Sect. 7.3). Moreover, a study on the initialization phases for both reduced AEGIS-128 and Tiaoxin-346 has been reported recently [119]. To further increase the resistance against the reported attacks, how to place the nonce and the key at the initial state is carefully chosen in our scheme.

7.1.2.3 Performance The encryption/decryption speed of Rocca is dramatically improved compared with other AES-based encryption schemes. Rocca is more than three and four times faster than SNOW-V and SNOW-V-GCM, respectively, i.e. the speed reaches 215 and 178 Gbps, respectively. Compared to other schemes with 256-bit key, Rocca is more than five times faster than AEGIS-256 and more than three times faster than AES-256-GCM in our evaluations (See Sect. 7.5 and Sect. 7.5.1). Moreover, Rocca is also faster than AEGIS-128, AEGIS-128L, and Tiaoxin-346 even though Rocca provides a higher security level. To the best of our knowledge, Rocca is the first dedicated cryptographic algorithm targeting 6G systems and we hope it can inspire future designs.

7.1.3 Organization

We first present the specification of Rocca in Sect. 7.2. Then, we describe the design rationale, such as the general construction based on AES-NI, criteria for performance and security, and how to find efficient round functions in Sect. 7.3. In Sect. 7.4, we provide the details of security evaluations against possible attacks on Rocca. Sect. 7.5 shows our software implementation results. Finally, we conclude this section in Sect. 7.6.

7.2 Preliminaries

In this section, the notations and the specification of our designs will be described.

7.2.1 Notations

The following notations will be used in Sect. 7. Throughout Sect. 7, a block means a 16-byte value. For the constants Z_0 and Z_1, we utilize the same ones as Tiaoxin-346 [137].

1. S: The state of Rocca, which is composed of 8 blocks, i.e. $S = (S[0], S[1], \ldots, S[7])$, where $S[i]$ $(0 \leq i \leq 7)$ are blocks and $S[0]$ is the first block.

2. Z_0: A constant block defined as $Z_0 =$ `428a2f98d728ae227137449123ef65cd`.

3. Z_1: A constant block defined as $Z_1 =$ `b5c0fbcfec4d3b2fe9b5dba58189dbbc`.

4. $\mathsf{AES}(X, Y)$: One AES round applied to the block X, where the round constant is Y, as defined below:

$$\mathsf{AES}(X, Y) = (\text{MixColumns} \circ \text{ShiftRows} \circ \text{SubBytes}(X)) \oplus Y,$$

   where MixColumns, ShiftRows and SubBytes are the same operations as defined in AES.

5. $A(X)$: The AES round function without the constant addition operation, as defined below:

$$A(X) = \text{MixColumns} \circ \text{ShiftRows} \circ \text{SubBytes}(X),$$

6. $|X|$: The length of X in bits.

7. 0^l: A zero string of length l bits.

8. $X||Y$: The concatenation of X and Y.

9. $R(S, X_0, X_1)$: The round function used to update the state S.

7.2.2 The Round Update Function

The input of the round function $R(S, X_0, X_1)$ of Rocca consists of the state S and two blocks (X_0, X_1). If denoting the output by S^{new}, $S^{new} \leftarrow R(S, X_0, X_1)$ can be defined as follows:

$$\begin{aligned}
S^{new}[0] &= S[7] \oplus X_0, \\
S^{new}[1] &= \mathsf{AES}(S[0], S[7]), \\
S^{new}[2] &= S[1] \oplus S[6], \\
S^{new}[3] &= \mathsf{AES}(S[2], S[1]), \\
S^{new}[4] &= S[3] \oplus X_1, \\
S^{new}[5] &= \mathsf{AES}(S[4], S[3]), \\
S^{new}[6] &= \mathsf{AES}(S[5], S[4]), \\
S^{new}[7] &= S[0] \oplus S[6].
\end{aligned}$$

The corresponding illustration can be referred to Figure 7.1.

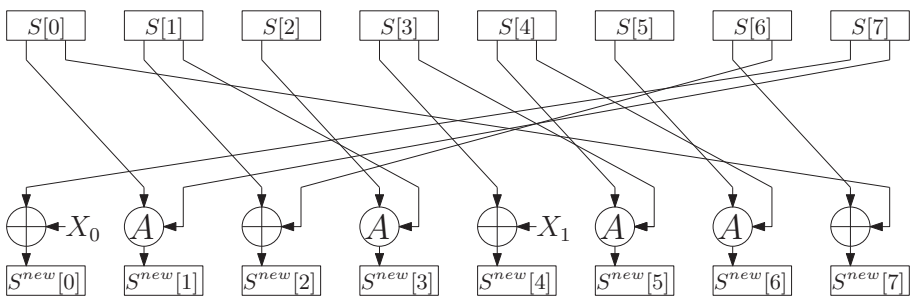


Figure 7.1: Illustration of the round function

7.2.3 Specification of Rocca

Rocca is an authenticated-encryption with associated-data scheme composed of four phases: initialization, processing the associated data, encryption and finalization. The input consists of a 256-bit key $K_0||K_1 \in \mathbb{F}_2^{128} \times \mathbb{F}_2^{128}$, a 128-bit nonce N, the associated data AD and the message M. The output is the corresponding ciphertext C and a 128-bit tag T. Define $\overline{X} = X||0^l$ where l is the minimal non-negative integer such that $|\overline{X}|$ is a multiple of 256. In addition, write X as $X = X_0||X_1||\ldots||X_{\frac{|X|}{256}-1}$ with $|X_i| = 256$. Further, X_i is written as $X_i = X_i^0||X_i^1$ with $|X_i^0| = |X_i^1| = 128$.

7.2.3.1 Initialization. First, (N, K_0, K_1) is loaded into the state S in the following way:

$$S[0] = K_1, S[1] = N, S[2] = Z_0, S[3] = Z_1,$$
$$S[4] = N \oplus K_1, S[5] = 0, S[6] = K_0, S[7] = 0.$$

Here, two 128-bit constants Z_0 and Z_1 are encoded as 16-byte little endian words and loaded into $S[2]$ and $S[3]$ respectively. Then, 20 iterations of the round function $R(S, Z_0, Z_1)$ is applied to the state S. After 20 iterations of the round function, two 128-bit keys are XORed with the state S in the following way;

$$S[0] = S[0] \oplus K_0,$$
$$S[4] = S[4] \oplus K_1.$$

7.2.3.2 Processing the associated data. If AD is empty, this phase will be skipped. Otherwise, AD is padded to $\overline{AD}$ and the state is updated as follows:

for $i = 0$ to $d - 1$
  $R(S, \overline{AD}_i^0, \overline{AD}_i^1)$,
end for

where $d = \frac{|\overline{AD}|}{256}$.

7.2.3.3 Encryption. The encryption phase is similar to the phase to process the associated data. If M is empty, the encryption phase will be skipped. Otherwise, M is first padded to $\overline{M}$ and then $\overline{M}$ will be absorbed with the round function. During this procedure, the ciphertext C is generated. If the last block of M is incomplete and its length is b bits, i.e. $0 < b < 256$, the last block of C will be truncated to the first b bits. A detailed description is shown below:

for $i = 0$ to $m - 1$
  $C_i^0 = \mathsf{AES}(S[1], S[5]) \oplus \overline{M}_i^0$,
  $C_i^1 = \mathsf{AES}(S[0] \oplus S[4], S[2]) \oplus \overline{M}_i^1$,
  $R(S, \overline{M}_i^0, \overline{M}_i^1)$,
end for

where $m = \frac{|\overline{M}|}{256}$.

7.2.3.4 Finalization. After the above three phases, two 128-bit keys K_0 and K_1 are first XORed with the state S in the following way;

$$S[0] = S[0] \oplus K_0,$$
$$S[4] = S[4] \oplus K_1.$$

Then, the state S will again pass through 20 iterations of the round function $\mathsf{R}(S, |AD|, |M|)$ and then the tag is computed in the following way:

$$T = \bigoplus_{i=0}^{7} S[i].$$

The length of associated data and message is encoded as 16-byte little endian word and stored into $|AD|$ and $|M|$, respectively.

A formal description of Rocca can be seen in Algorithm 1 and the corresponding illustration is shown in Figure 7.2.

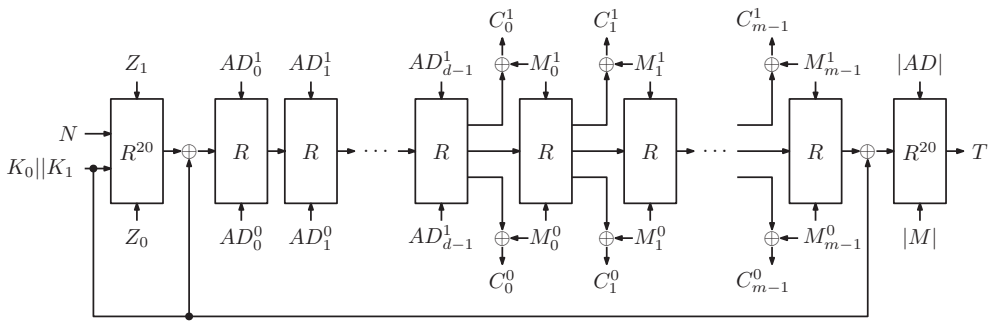


Figure 7.2: The procedure of Rocca

7.2.3.5 A raw encryption scheme. If the phases of processing the associated data and finalization are removed, a raw encryption scheme is obtained.

7.2.3.6 Test Vectors This section gives three test vectors of Rocca. The least significant byte of the vector is shown on the left and the first 256-bit value (in plaintext and ciphertext) is shown on the first line.

7.2.3.7 Security claims. Rocca provides 256-bit security against key-recovery and 128-bit security against distinguishing and forgery attacks in the nonce-respecting setting[13]. We do not claim its security in the related-key and known-key settings.

The message length for a fixed key is limited to at most 2^{128} and we also limit the number of different messages that are produced for a fixed key to be at most 2^{128}. The length of associated data of a fixed key is up to 2^{64}.

[13]We updated the claimed security of distinguishing attacks from the ToSC version [154] for the following reasons. The most well-known and popular distinguishing attack on the keystream seems to be the linear attack. Such a distinguishing attack often requires a large number of plaintexts. If the data complexity exceeds the time complexity to find the key with Grover's algorithm, we view such an attack as invalid in the quantum setting. Therefore, regarding the distinguishing attack, we only claim 128-bit security in the quantum setting and a meaningful distinguishing attack in the classical setting should have data complexity below 2^{128}.

Algorithm 1 The specification of Rocca

```
procedure RoccaEncrypt(K_0, K_1, N, AD, M)
    S ← Initialization(N, K_0, K_1)
    if |AD| > 0 then
        S ← ProcessAD(S, \overline{AD})
    if |M| > 0 then
        S ← Encryption(S, \overline{M}, C)
        Truncate C
    T ← Finalization(S, |AD|, |M|, K_0, K_1)
    return (C, T)
procedure RoccaDecrypt(K_0, K_1, N, AD, C, T)
    S ← Initialization(N, K_0, K_1)
    if |AD| > 0 then
        S ← ProcessAD(S, \overline{AD})
    if |C| > 0 then
        S ← Decryption(S, \overline{C}, M)
        Truncate M
    if T = Finalization(S, |AD|, |C|, K_0, K_1) then
        return M
    else
        return ⊥
procedure Initialization(N, K_0, K_1)
    (S[0], S[1], S[2], S[3]) ← (K_1, N, Z_0, Z_1), (S[4], S[5], S[6], S[7]) ← (N ⊕ K_1, 0, K_0, 0)
    for i = 0 to 19 do
        S ← R(S, Z_0, Z_1)
    (S[0], S[4]) ← (S[0] ⊕ K_0, S[4] ⊕ K_1)
    return S
procedure ProcessAD(S, AD)
    d ← |AD| / 256
    for i = 0 to d − 1 do
        S ← R(S, AD_i^0, AD_i^1)
    return S
procedure Encryption(S, M, C)
    m ← |M| / 256
    for i = 0 to m − 1 do
        C_i^0 ← AES(S[1], S[5]) ⊕ M_i^0, C_i^1 ← AES(S[0] ⊕ S[4], S[2]) ⊕ M_i^1
        S ← R(S, M_i^0, M_i^1)
    return S
procedure Decryption(S, M, C)
    c ← |C| / 256
    for i = 0 to c − 1 do
        M_i^0 ← AES(S[1], S[5]) ⊕ C_i^0, M_i^1 ← AES(S[0] ⊕ S[4], S[2]) ⊕ C_i^1
        S ← R(S, M_i^0, M_i^1)
    return S
procedure Finalization(S, |AD|, |M|, K_0, K_1)
    (S[0], S[4]) ← (S[0] ⊕ K_0, S[4] ⊕ K_1)
    for i = 0 to 19 do
        S ← R(S, |AD|, |M|)
    T ← 0
    for i = 0 to 7 do
        T ← T ⊕ S[i]
    return T
```

Table 7.1: Test vectors.

K	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
N	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
M	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C	15 89 2f 85 55 ad 2d b4 74 9b 90 92 65 71 c4 b8 c2 8b 43 4f 27 77 93 c5 38 33 cb 6e 41 a8 55 29
	17 84 a2 c7 fe 37 4b 34 d8 75 fd cb e8 4f 5b 88 bf 3f 38 6f 22 18 f0 46 a8 43 18 56 50 26 d7 55
T	cc 72 8c 8b ae dd 36 f1 4c f8 93 8e 9e 07 19 bf
K	01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
N	01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
A	01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
M	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C	60 9b 60 18 7e ae 09 ee 0d df 95 af 40 86 e7 66 32 5c 17 03 26 c2 9d 91 b2 4d 71 4f ec f3 85 fd
	09 87 f8 20 cb f7 ca bb 11 52 43 2c 6d 60 5a 8e c5 7e af 08 2b b4 e7 2b 9e 54 5e 5c 59 01 3d af
T	74 0e 79 c5 e5 9b d2 91 5f da 57 9d 51 7a c4 1d
K	01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef
N	01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef
A	01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef 01 23 45 67 89 ab cd ef
M	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C	44 c7 46 1b 6c 13 08 79 da 05 de 5e f8 8e da 35 91 a2 a7 ae ff 91 ef d3 ac 60 3b 28 e0 57 61 09
	5d d8 a1 87 bf 57 8b 5f a6 04 ed 5e 61 f0 3c 0b 81 91 05 c3 6f 6a b7 97 59 7b 67 e3 80 3b 6a 04
T	1e 5e fe fa 86 df b6 e5 5a d5 bb 9d bd a6 98 e5

K: Master key N: Nonce A: associated data M: Plaintext C: Ciphertext T: tag

7.3 Design Rationale

7.3.1 General Construction

SIMD instruction. The prime design goal of Rocca is to meet the requirements of processing/-transmission speed for 6G applications, namely more than 100 Gbps [117]. In order to realize fast encryption/decryption speed (hereafter, we simply call "speed") on a pure software environment, we take full advantage of the SIMD instructions and the AES-NI, both of which are equipped on most of modern CPUs. The SIMD instructions contains some fundamental instructions such as XOR and AND, and can execute them by 32/64/128-bit units as one instruction, where the AES-NI is a special set of the SIMD instructions, which is first rolled out by Intel [60] and available on modern processors. The AES-NI can execute AES about 10 times faster than non-AES-NI in parallelizable modes such as CTR mode. In Sect. 7, we utilize on aesenc, which is one of instruction sets of AES-NI, and performs one regular (not the last) round of AES on an input state S with a subkey K:

$$\mathsf{aesenc}(S, K) = (\mathsf{MixColumns} \circ \mathsf{ShifRows} \circ \mathsf{SubBytes}(S)) \oplus K.$$

The execution speed of these instructions can be evaluated by *latency* and *throughput*, where latency is the number of clock cycles required to execute a single instruction and throughput is the required number of clock cycles before the same instruction to be executed. It is important when considering the parallel execution. Table 7.2 shows latency and throughput of aesenc [148] in each architecture. Among existing architectures, we focus the latest architecture Intel Ice-Lake series that has the fastest AES-NI whose latency and throughput of aesenc are 3 and 0.5, respectively. Figure 7.3 illustrates an example of the process in the parallel execution of aesenc for Intel Ice-lake whose latency and throughput are 3 and 0.5[14], respectively.

Employing one AES round as an underlying component for future designs has a great merit for performance compared to employing other cryptographic primitives. Many software and libraries support AES-NI natively, e.g OpenSSL. Thus, it seems to be very reasonable that devices connected to 6G services will still support such instructions. SNOW-V also takes advantage of AES-NI for the same reason.

Permutation-based Structure. As a reference point, we consider a stream cipher SNOW-V,

[14]Throughput 0.5 means that there are two ports for aesenc with throughput 1.

Table 7.2: Latency and throughput of aesenc for some architectures by Intel and AMD referred by [148].

Vendor	Architecture	Latency	Throughput
Intel	Sky-lake	4	1
	Kaby-lake	4	1
	Coffee-lake	4	1
	Cannon-lake	4	0.5
	Cascade-lake	4	1
	Comet-lake	unknown	unknown
	Ice-lake	3	0.5
AMD	Zen +	4	0.5
	Zen 2	4	0.5

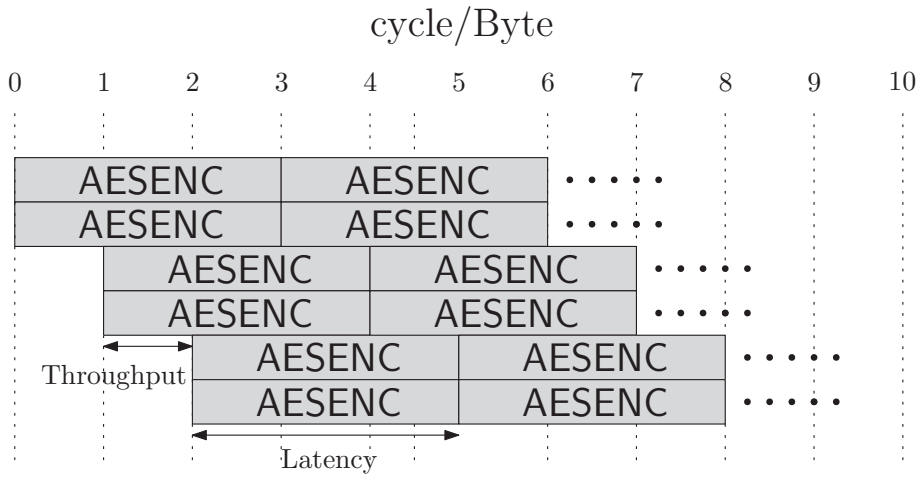


Figure 7.3: The process of aesenc for Intel Ice-lake.

which is designed for 5G applications. SNOW-V is based on linear feedback shift register (LFSR) and Finite State Machine(FSM) with AES-based round functions. As discussed in Section 7.1, if we follow this design strategy, we need to accelerate the performance approximately at least three times faster than SNOW-V to achieve the required performance of 100 Gbps. Thus, we decide to choose other design strategies based on AES round functions.

Specifically, we focus on AEGIS family [175] and Tiaoxin-346 [137], which are permutation-based authenticated encryption schemes using AES round functions and submitted to CAESAR competition [2]. These allow a full parallelization and can achieve the outstanding speed compared to AES-CTR.

However, as it has been pointed out that there exists a linear bias in the ciphertext blocks for AEGIS-256 [129], it seems insecure to adopt the similar quadratic boolean function to generate the ciphertexts, especially for the purpose to reach 256-bit security. This fact motivates us to design different ways to generate the ciphertext blocks and finally involving 1 AES round function into generating each ciphertext block is chosen. Such a way is efficient due to the parallel calls to AES-NI. Moreover, a study on the initialization phases for both reduced AEGIS-128 and Tiaoxin-346 has been reported recently [119]. To further increase the resistance against the reported attacks, how to place the nonce and the key at the initial state is carefully chosen in our scheme, which is little discussed in AEGIS and Tiaoxin-346.

Efficient AES-Based Round Function. Round functions of AEGIS family [175] and Tiaoxin-346 [137] consist of the 128-bit XOR operation and one AES round that is executed by the processor instruction aesenc. Jean and Nikolić have generalized the way to construct efficient

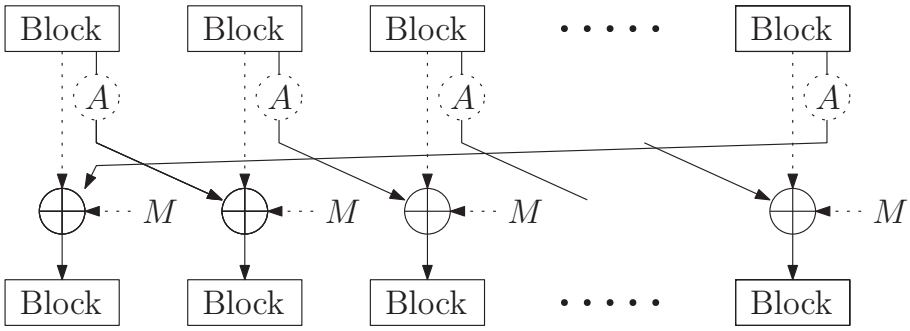


Figure 7.4: The general construction considered of the round function in [103]. Dash lines mean that it can be possible to be absent or present in the design.

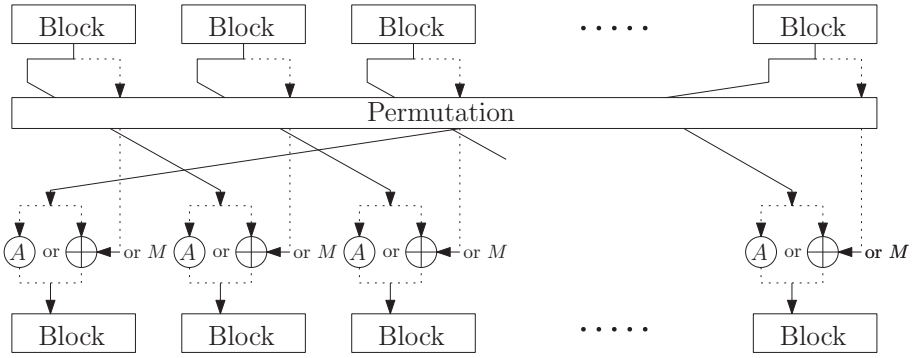


Figure 7.5: General construction of the round function. Dash lines mean that it can be possible to be absent or present in the design.

round functions using only the one AES round (aesenc) and 128-bit XOR and have found several more efficient candidates [103]. Figure 7.4 shows the general construction of the round function considered in [103].

To push the limitation further of efficiency of their structures, we explore a new class of AES-based structures shown in Fig 7.5. Compared to the structures considered by Jean and Nikolić results [103], our constructions remove the case of applying both aesenc and XOR to each block in a cascade way for one round to minimize the critical path of one round. Specifically, we only consider the case of applying only either aesenc or 128-bit XOR to each block in one round, where aesenc takes a state block or message block as input of AddRoundKey and 128-bit XOR takes state block or message block as inputs, respectively as shown in Figure 7.5.

Moreover, we apply a block permutation to state blocks, which was not considered by Jean and Nikolić (See Fig 7.4). This sufficiently increases the number of possible candidates. Indeed, as described in later section, it enables us to find more efficient constructions than Jean and Nikolić's results, which is not covered by their target classes. It should be emphasized that executing the block permutation in register size is a cost-free operation, that is, the permutation only changes the order of blocks. More strictly, a permutation needs some temporary registers. However, these registers almost do not affect the speed if the total number of registers used in process of the scheme is lower than 16, which is the total number of xmm-registers equipped in almost all modern CPUs. Hence, applying a block permutation does not affect the speed of the round function. For a block that will be inputted into aesenc or XOR, we use one-block right rotation as in [103].

7.3.2 Criteria for Performance and Security

For designing efficient round functions, we need to choose several parameters such as the number of aesencs, the number of inserted message blocks, and a block permutation for our structure in Fig. 7.5. We clarify requirements of performance and security for target applications to choose these parameters.

7.3.2.1 Requirements for Performance. To theoretically estimate speed, we utilize a metric called *rate*, which is proposed by Jean and Nikolić [103].

Definition 17 (Rate [103]) *The rate p of a design is the number of* AES *rounds (calls to* aesenc*) used to process a 128-bit message.*

For our general construction of Fig 7.5, the *rate* p is estimated as a ratio of (# of aesencs)/(# of the inserted 128-bit messages) in one round. Since a smaller *rate* leads to more efficient design [103], we should design the round function that have as small *rate* as possible. The *rate* is the most important parameter for speed.

The number of aesenc in one round is also important factor to maximize the efficiency. Jean and Nikolić claim that the number of aesenc in one round should be close to (latency)/(throughput) ratio [103] for the efficient design, e.g. if the latency and throughput of aesenc are respectively 3 and 0.5, the number of aesenc should be 6 in one round. The reason is when the number of aesencs is less than a (latency)/(throughput) ratio, there are empty cycles in process of aesenc. On the other hand, if the number of aesencs is the same as (latency)/(throughput) ratio, there is no empty cycles as shown in Figure 7.3. Since our target architecture is Ice-lake, the number of aesenc in a round should be 6.

Another important factor related to speed is the number of blocks of round functions, namely the state size. Smaller state size significantly improves the efficiency because it can reduce registers used for encryption and makes a whole process of encryption easier. We experimentally confirmed that reducing the number of blocks leads to increasing speed when the *rate* is the same. Table 7.3 shows our experimental result that compares three types of round functions of the *rate* 2 with the number of blocks of 8, 9, and 10, each of which is measured on Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30GHz with 16 GB RAMs. Details of these round functions are given in Fig 7.6, 7.7, and 7.8. The round function whose # of blocks is 8 is the same as the one of Rocca. Other 2 round functions whose # of blocks is 9 and 10 are the simple extended version of that.

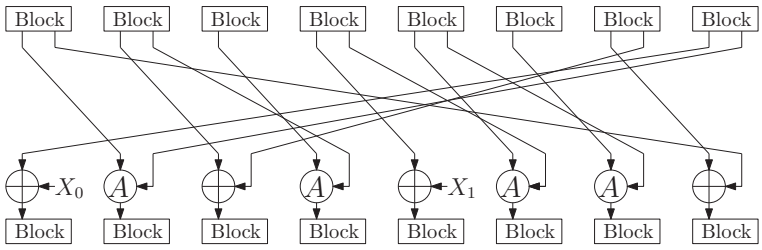


Figure 7.6: The round function whose # of blocks is 8.

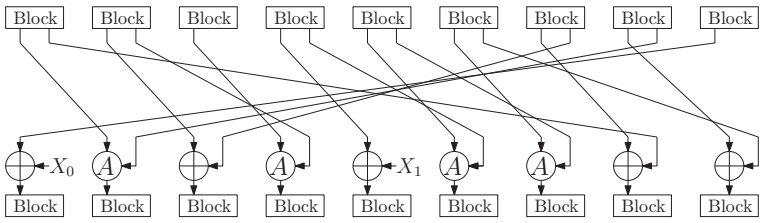


Figure 7.7: The round function whose # of blocks is 9.

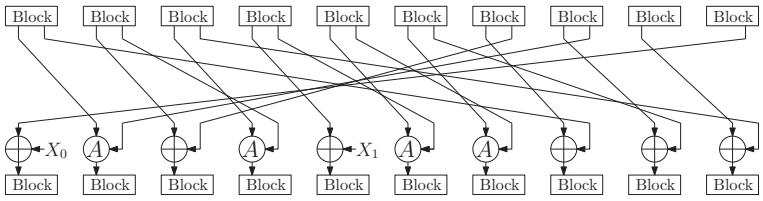


Figure 7.8: The round function whose # of blocks is 10.

Besides, a smaller state size is a preferable feature to be deployed in wider classes of devices with keeping the efficiency. It is because this, that some CPUs, such as ones from AMD, do not support the large size register like AVX512, and the process requiring the use of many registers tends to become more complicated on these CPUs. Since the number of blocks of SNOW-V, which is our reference point, is 7, the state size should be competitive.

Table 7.3: Comparison of the performance of the round function having different number of blocks at the same *rate*.

# of blocks	Speed (in cycle/Byte)	rate
8	0.126717	2
9	0.147397	2
10	0.155584	2

7.3.2.2 Requirements for Security. Since evaluating the resistance to all possible attacks for all possible candidates is practically infeasible, we focus on the security against the forgery attack by the internal collision as a criteria of security when finding candidates, as with [103]. Especially, we impose the 128-bit security against the forgery attack on our design, i.e. our security requirement is that there are no internal collisions with a probability more than 2^{-128}. Through Sect. 7, "forgery attacks" is meant to be a universal forgery in the nonce-respecting setting.

To evaluate the probability of the internal collision, we search the lower bound for the number of active S-boxes by a Mixed Integer Linear Programming (MILP) solver [133]. Since the maximum probability of an S-box is 2^{-6}, it is sufficient to guarantee the security against internal collisions if there are 22 active S-boxes, as it gives $2^{(-6\times 22)} < 2^{-128}$ as an estimate of differential probability. For the security against other possible attacks, we evaluate after designing a whole design, and it will be described in Sect. 7.4.

7.3.2.3 Summary of Our Criteria. Requirements for AES-based round function are as follows.

For speed.

Requirement 1. The lowest *rate* round function as possible that leads to faster speed.

Requirement 2. The number of aesencs in one round is close to 6.

Requirement 3. A round function with a smaller number of blocks (around 7).

For security.

Requirement 4. 128-bit security to the forgery attack by internal collision, i.e. the lower bound of active S-boxes is 22.

For comparison, Table 7.4 shows parameters of the round function in the AEGIS [175] family, Tiaoxin-346 [137] and structure by Jean and Nikolić [103].

Table 7.4: Round functions of AEGIS family and Tiaoxin-346

Primitive	# of aesenc	# of blocks	# of inserted message blocks	rate
AEGIS-128	5	5	1	5
AEGIS-256	6	6	1	6
AEGIS-128L	8	8	2	4
Tiaoxin-346	6	13	2	3
[103]	6	12	3	2

7.3.3 Finding Efficient Structures

We choose several parameters such as the number of aesencs, the number of inserted message blocks, and a block permutation to meet requirements given in Sect. 7.3.2. The number of possible candidates is estimated as $s! \times \binom{s}{a} \times \binom{s}{m}$ candidates where s, a, and m are # of blocks, # of aesenc, and # of message blocks, respectively. For example, it reaches $2^{35.00}$ candidates when $s = 10$, $a = 4$, and $m = 2$.

7.3.3.1 Our Approach. According to Table 7.4, the most efficient design is Jean and Nikolić's structure whose $rate$ is 2. However, their state size is quite large for our requirement. In our experiments, the round functions with a smaller $rate$ require a larger number of blocks to meet the security requirement. Indeed, we cannot find any structure of $rate$ 2 and less than 12 internal blocks by Jean and Nikolić's constructions (Fig.7.4) [103]. To address it, our approach is as follow.

- To expand possible candidates while keeping efficiency, we introduce a block permutation to state blocks in the round function, while Jean and Nikolić did not consider any permutation. It should be emphasized that executing the block permutation in register size is a cost-free operation.
- To further improve the efficiency, we focus on the structure in which each block in one round is applied only either aesenc or XOR to minimized the critical path of the round function.

7.3.3.2 Search Targets. When the number of inserted message blocks is m, the number of aesencs in one round should be $(6 - m)$ to satisfy requirement 2 as m aesenc is used for generating ciphertext blocks for our design to the resistance to the linear bias (details in Section 3.5). Considering requirement 1 ($rate$ = 2), the only choice of m is 2, thus the number of aesencs is 4. Following requirement 3, we consider the case where # of blocks are from 6 to 8. Besides, we consider the case where $rate$ = 1.5 that can not satisfy requirement 2, because the low $rate$ round function might be possible to more efficient even if it does not meet requirement 2. Table 7.5 shows our candidates of the round function.

Table 7.5: Candidates of round functions which we search.

Round function	# of aesenc	# of blocks	# of message blocks	rate	# of candidates	# of searched candidates
Candidates-1	4	6	2	2	$2^{17.30}$	ALL
Candidates-2	4	7	2	2	$2^{21.82}$	ALL
Candidates-3	4	8	2	2	$2^{26.23}$	$2^{19.93}$
Candidates-4	3	6	2	1.5	$2^{17.72}$	ALL
Candidates-5	3	7	2	1.5	$2^{21.82}$	ALL
Candidates-6	3	8	2	1.5	$2^{25.91}$	$2^{19.93}$

We evaluate the lower bounds for the number of active S-boxes for Candidate-1, 2, 3, 4, 5, and 6 by a MILP solver. We can conduct exhaustive searches for Candidates-1, 2, 4, and 5 while exhaustive searches for Candidates-3 and 6 are infeasible due to too large candidates that reach $2^{26.23}$ and $2^{25.91}$ for Candidates-3 and 6, respectively. Thus, we randomly search $2^{19.93}$ candidates for both Candidate-3 and 6.

7.3.3.3 Results. As a result of an exhaustive search over Candidates-1, 2, 4, and 5, there are no round functions that satisfy the requirement 4. For candidates-6, we could not find round functions meeting requirement 4 either. For Candidates-3, we found that 100 out of $2^{19.93}$ candidates ensure active S-boxes of ≥ 22. We then evaluate a diffusion property for these 100 candidates. Then we find 22 out of 100 candidates achieve the full diffusion after 7 rounds in nibble-wise while round functions of AEGIS-128, AEGIS-256, AEGIS-128L, and [103] require 7, 8, 10, and 12 rounds for the full diffusion, respectively, and the one of Tiaoxin-346 never achieve the full diffusion as the state consists of three independent chucks.

We finally choose the round function shown in Fig 7.1 as the one of Rocca, which ensures active S-boxes of 24 that is the largest number of active S-boxes among 22 candidates. This evaluation requires about 23 days on three computers equipped with 48/64/64 cores and 256/256/256 GB RAMs.

Table 7.6 compares the speed of round functions of Rocca and other primitives, where speed is estimated as the average value of the round function executed 1000000 times with 64kB messages on Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30GHz with 16 GB RAMs. Our round function is the fastest one and the number of blocks is smaller than ones whose $rate$ is 2 or 3.

It should be mentioned that the comparison of the speed of round functions does not always reflect directly to the speed of the whole design. This is because that the overhead of the ciphertext generation depends on the structure of the round function, especially the empty cycle in process of XOR/aesenc.

Table 7.6: Speed (in cycles / Byte) of round functions of Rocca, AEGIS-128, AEGIS-128L, AEGIS-256, Tiaxion-346, and JN16 (not include a generation part of a ciphertext).

Primitive	Speed (in cycles / Byte)	# of blocks	rate
AEGIS-128	0.384482	5	5
AEGIS-256	0.388125	6	6
AEGIS-128L	0.191072	8	4
Tiaoxin-346	0.192413	13	3
[103]	0.140433	12	2
Rocca	0.124609	8	2

7.3.4 Loading the Nonce and Key

It has been pointed by Liu et al. that there is one useless round in Tiaoxin-346 by expressing the internal states in terms of the nonce and the key at the initialization phase [119]. The main reason is that the nonce and the key are not well diffused, i.e. after a certain number of rounds, the

internal state can be expressed in terms of $A(N)$ and the key. To avoid it in Rocca, we carefully investigate how to place the nonce and the key.

In Rocca, the initial state is loaded as follows:

$$S[0] = K_1, S[1] = N, S[2] = Z_0, S[3] = Z_1,$$
$$S[4] = N \oplus K_1, S[5] = 0, S[6] = K_0, S[7] = 0.$$

After one-round update, the state $(S[0], \ldots, S[7])$ becomes:

$$S[0] = Z_0, S[1] = A(K_1), S[2] = \underline{N \oplus K_0}, S[3] = \underline{N \oplus A(Z_0)},$$
$$S[4] = 0, S[5] = \underline{A(N \oplus K_1) \oplus Z_1}, S[6] = \underline{N \oplus K_1}, S[7] = K_0 \oplus K_1.$$

It can be observed that N is xored with K_0 and K_1, respectively. Moreover, N is involved in the expressions of each state block in a very different way, which can avoid the useless rounds and, at the same time, strengthen the resistance against the key-recovery attacks applied to round-reduced AEGIS-128 and Tiaoxin-346 as described in [119]. Further evidence can be seen from the expressions of the state blocks after 3 rounds of update, as shown below:

$$\begin{aligned}
S[0] &= \underline{N \oplus K_1}, \\
S[1] &= A(K_0 \oplus K_1 \oplus Z_0) \oplus Z_0 \oplus N \oplus K_1, \\
S[2] &= A(Z_0) \oplus K_0 \oplus K_1 \oplus A(A(\underline{N \oplus K_1}) \oplus Z_1), \\
S[3] &= A(\underline{A(K_1) \oplus N \oplus K_1}) \oplus A(Z_0) \oplus K_0 \oplus K_1, \\
S[4] &= A(\underline{N \oplus K_0}) \oplus A(K_1) \oplus Z_1, \\
S[5] &= A(\underline{N \oplus A(Z_0) \oplus Z_1}) \oplus A(\underline{N \oplus K_0}) \oplus A(K_1), \\
S[6] &= A(\underline{N \oplus A(Z_0)}) \oplus N \oplus A(Z_0) \oplus Z_1, \\
S[7] &= K_0 \oplus K_1 \oplus Z_0 \oplus A(A(\underline{N \oplus K_1}) \oplus Z_1).
\end{aligned}$$

7.3.5 Generating the Ciphertext Blocks

In both AEGIS and Tiaoxin-346, each ciphertext block is computed based on a simple quadratic boolean function in terms of the several internal state blocks where the number of AND operations is 1. However, such a way to generate the output seems to be insecure against the statistical attack proposed by [129], especially for the scheme targeting 256-bit security.

At the initial design phase, we tried many possible combinations to compute each ciphertext block with a similar quadratic boolean function. However, with the MILP-based method [80] to automatically evaluate the security against this statistical attack, the lower bound for the time complexity is always below 2^{128}, which is far smaller than 2^{256}. Therefore, new strategies are essential for Rocca.

The basic idea is to utilize a complex nonlinear function and finally the AES round function is chosen as the only nonlinear function. Due to the parallel way to perform the AES round function, such a way is indeed rather efficient and can simultaneously strengthen the security of our scheme. To reduce the overall overheads, computing each ciphertext block only utilizes 1 aesenc.

The basic principle to choose the state blocks to compute the ciphertext is that the state blocks $(S[0], S[2], S[4], S[5])$ passing through the AES round function in the round updated function should be involved, which can increase the number of active S-boxes in the first round. In addition, we expect that they should be processed in a different way from that in the round update function. Intuitively, this can prevent the ciphertext blocks from being related to the updated internal state blocks.

Moreover, as $(S[4], S[5])$ passes through the AES round function in the round update function and the two state blocks are next to each other, considering the fact that several rounds are needed, it is better to choose additional state blocks from $(S[0], S[1], S[2], S[3], S[4])$, which will be shifted to $(S[4], S[5])$ after some rounds. A detailed study of the security of our choice can be found in the following section.

We emphasize that the overhead of executing these two aesencs is few since we can assign them into empty cycles of aesenc in the round function.

7.4 Security Evaluation

7.4.1 Differential Attack

The differential attack is one of the possible attacks on the initialization phase of Rocca. Specifically, the differences in the $nonce$ (and key) can propagate to the ciphertext. If there is a differential characteristic with a high probability, it can be exploited for the differential attack. Hence, we can compute the lower bound for the number of active S-boxes in the initialization phase to evaluate the resistance against the differential attack. To compute the lower bound, we utilize a MILP-aided method proposed by Mouha et al. [133] and focus on the byte-wise truncated differences. We evaluate it in both the single-key setting where differences can only be injected into the $nonce$ and the related-key setting where differences can be injected into the key and $nonce$.

Table 7.7 shows the lower bounds for the number of active S-boxes up to 14 rounds in the single-key setting and up to 11 rounds in the related-key setting in the initialization phase. Since the maximal differential probability of the S-box of AES is 2^{-6}, it is sufficient to guarantee the security against differential attacks if there are 43 active S-boxes, as it gives $2^{(-6\times 43)} < 2^{-256}$ as an estimate of the differential probability. As shown in Table 7.7, there are 54 active S-boxes over 6 rounds in the single-key setting and 44 active S-boxes over 7 rounds in the related-key setting in the initialization phase. It should be emphasized that we do not claim the security in the related-key setting, although we evaluated the number of active S-boxes in the related-key setting.

Since there is a large security margin, we expect that Rocca can resist against differential attacks in the initialization phase.

Table 7.7: The lower bound for the number of active S-boxes in the initialization phase where AS_{sk} and AS_{rk} mean an active S-box in the single-key setting and in the related-key setting, respectively.

Rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# of AS_{sk}	1	6	9	30	38	54	62	82	85	93	100	104	111	115
# of AS_{rk}	0	1	2	11	21	36	44	48	68	73	79	-	-	-

7.4.2 Forgery Attack

It has been shown in [137] that the forgery attack is a main threat to the constructions like Tiaoxin-346 and AEGIS as only one-round update is used to absorb each block of associated data and message. Such a concern has been taken into account in our design phase, as reported in Sect. 7.3.

Specifically, in the forgery attack, the aim is to find a differential trail where the attackers can arbitrarily choose differences at the associated data and expect that such a choice of difference can lead to a collision in the internal state after several number of rounds. The resistance against this attack vector can be efficiently evaluated with an automatic method [133]. As Rocca is based on the AES round function, it suffices to prove that the number of active S-boxes in such a trail is larger than 22 as the length of the tag is 128 bits. With the MILP-based method, it is found that the lower bound is 24. Consequently, Rocca can provide 128-bit security against the forgery attack.

7.4.3 Integral Attack

One of the most efficient attacks on round-reduced AES is integral attacks. Recently, Liu et al. presented some attacks [119] on round-reduced AEGIS-128 and Tiaoxin-346 based on the integral distinguisher on 4-round AES. To understand the security of our construction, it is necessary to evaluate the resistance against integral attacks. Similar to [119], the internal state will be first expressed in terms of the initial state and then we study the expressions.

For simplicity, denote the state after r iterations of the round function at the initialization phase by S_r. In addition, when writing the expressions, we omit the constants and use $A(X)$ to represent that X passes through one AES round, i.e. $A(X)$ can represent $A(X \oplus \epsilon)$ where ϵ is a 128-bit constant. In this way, the internal state S_4 can be expressed as follows:

$$\begin{aligned}
&S_4[0] = A(A(N)), S_4[1] = A(N) \oplus A(A(N)),\\
&S_4[2] = A(N), S_4[3] = A(A(A(N))) \oplus N,\\
&S_4[4] = A(N), S_4[5] = A(A(N)) \oplus A(N),\\
&S_4[6] = A(A(N) \oplus A(N)) \oplus A(N), S_4[7] = A(N).
\end{aligned}$$

As our construction can provide 256-bit security, it is necessary to evaluate the case when N traverses all the 2^{128} possible values under the same 256-bit key. According to [119], some terms in the expressions can be eliminated by adding proper conditions and the expressions can be simplified. However, according to the expression of $S_4[3]$, when N takes all the possible values, it is impossible that $S_4[3]$ will also take all the 2^{128} possible values. In other words, the multiset of $S_4[3]$ tends to be unstructured. Therefore, by considering the propagation of $S_4[3]$ and the way to compute the ciphertext, we believe that 20 rounds are sufficient to resist against integral attacks.

On the other hand, consider the expressions for S_6, as shown below:

$$\begin{aligned}
S_6[0] &= A(A(N)) \oplus A(A(N) \oplus A(N)) \oplus A(N),\\
S_6[1] &= A(A(N)) \oplus A(A(N)) \oplus A(A(N)) \oplus A(A(N) \oplus A(N)) \oplus A(N),\\
S_6[2] &= A(A(A(N))) \oplus A(N) \oplus A(A(A(N)) \oplus A(N)) \oplus A(N),\\
S_6[3] &= A(A(N) \oplus A(A(N)) \oplus A(A(N) \oplus A(N)) \oplus A(N)) \oplus A(N))\\
&\quad \oplus A(A(A(N))) \oplus A(N),\\
S_6[4] &= A(A(N)) \oplus A(N) \oplus A(A(N)),\\
S_6[5] &= A(A(A(A(N))) \oplus N) \oplus A(A(N)) \oplus A(N) \oplus A(A(N)),\\
S_6[6] &= A(A(A(N)) \oplus A(A(A(N))) \oplus N) \oplus A(A(A(N))) \oplus N,\\
S_6[7] &= A(N) \oplus A(A(A(N)) \oplus A(N)) \oplus A(N).
\end{aligned}$$

As

$$\begin{aligned}
S_8[0] \oplus S_8[4] &= S_6[0] \oplus S_6[6] \oplus A(S_6[2]) \oplus S_6[1] \oplus Z_0 \oplus Z_1,\\
S_8[1] &= A(S_6[7] \oplus Z_0) \oplus S_6[0] \oplus S_6[7],
\end{aligned}$$

it can be found that in the expressions of $A(S_8[1])$ and $A(S_8[0] \oplus S_8[4])$, N will pass through 5 AES rounds and there seems to be no way to add proper conditions to prevent N from passing through 5 AES rounds. Moreover, as N passes through 5 AES rounds in very different ways in $A(S_8[1])$ and $A(S_8[0] \oplus S_8[4])$, it is also impossible to prevent it by considering the sum $A(S_8[1]) \oplus A(S_8[0] \oplus S_8[4])$. Consequently, we further believe that 20 rounds are secure against integral attacks.

7.4.4 State-recovery Attack

Different from AEGIS and Tiaoxin-346, the output in our construction only involves a few state blocks, i.e. the attackers are able to know $A(S[1]) \oplus S[5]$ and $A(S[0] \oplus S[4]) \oplus S[2]$. As the internal state consists of 8 blocks and the output in each round only leaks 256-bit information, the attackers at least need to consider 4 consecutive rounds in order to recover the whole secret internal state.

7.4.4.1 Guess-and-determine attack. The guess-and-determine attack is a common tool to achieve state recovery. Consider four consecutive rounds at the encryption phase and denote the 4 internal states used to generate the ciphertexts by S_t, S_{t+1}, S_{t+2} and S_{t+3}, respectively. In this case, the attackers can compute

$$A(S_i[1]) \oplus S_i[5], A(S_i[0] \oplus S_i[4]) \oplus S_i[2],$$

where $t \leq i \leq t+3$.

Assuming the message blocks are all zero, we thus have

$$\begin{aligned}
A(S_{t+1}[1]) &= A(A(S_t[0]) \oplus S_t[7]), \\
S_{t+1}[5] &= A(S_t[4]) \oplus S_t[3], \\
A(S_{t+1}[0] \oplus S_{t+1}[4]) &= A(S_t[7] \oplus S_t[3]), \\
S_{t+1}[2] &= S_t[1] \oplus S_t[6],
\end{aligned}$$

$$\begin{aligned}
A(S_{t+2}[1]) &= A(A(S_{t+1}[0]) \oplus S_{t+1}[7]) \\
&= A(A(S_t[7]) \oplus S_t[0] \oplus S_t[6]), \\
S_{t+2}[5] &= A(S_{t+1}[4]) \oplus S_{t+1}[3] \\
&= A(S_t[3]) \oplus A(S_t[2]) \oplus S_t[1], \\
A(S_{t+2}[0] \oplus S_{t+2}[4]) &= A(S_{t+1}[7] \oplus S_{t+1}[3]) \\
&= A(S_t[0] \oplus S_t[6] \oplus A(S_t[2]) \oplus S_t[1]), \\
S_{t+2}[2] &= S_{t+1}[1] \oplus S_{t+1}[6] \\
&= A(S_t[0]) \oplus S_t[7] \oplus A(S_t[5]) \oplus S_t[4],
\end{aligned}$$

$$\begin{aligned}
A(S_{t+3}[1]) &= A(A(S_{t+1}[7]) \oplus S_{t+1}[0] \oplus S_{t+1}[6]) \\
&= A(A(S_t[0] \oplus S_t[6]) \oplus S_t[7] \oplus A(S_t[5]) \oplus S_t[4]), \\
S_{t+3}[5] &= A(S_{t+1}[3]) \oplus A(S_{t+1}[2]) \oplus S_{t+1}[1], \\
&= A(A(S_t[2]) \oplus S_t[1]) \oplus A(S_t[1] \oplus S_t[6]) \oplus A(S_t[0]) \oplus S_t[7], \\
A(S_{t+3}[0] \oplus S_{t+3}[4]) &= A(S_{t+1}[0] \oplus S_{t+1}[6] \oplus A(S_{t+1}[2]) \oplus S_{t+1}[1]), \\
&= A(A(S_t[5]) \oplus S_t[4] \oplus S_t[1] \oplus S_t[6] \oplus A(S_t[0])), \\
S_{t+3}[2] &= A(S_{t+1}[0]) \oplus S_{t+1}[7] \oplus A(S_{t+1}[5]) \oplus S_{t+1}[4], \\
&= A(S_t[7]) \oplus S_t[0] \oplus S_t[6] \oplus A(A(S_t[4]) \oplus S_t[3]) \oplus S_t[3].
\end{aligned}$$

Therefore, the attackers at least need to consider the following 1024 nonlinear boolean equa-

tions in terms of 1024 boolean variables $(S_t[0], \ldots, S_t[7])$ in order to recover the secret state:

$$
\begin{aligned}
\alpha_0 &= A(S_t[1]) \oplus S_t[5], \\
\alpha_1 &= A(S_t[0] \oplus S_t[4]) \oplus S_t[2], \\
\alpha_2 &= A(A(S_t[0]) \oplus S_t[7]) \oplus A(S_t[4]) \oplus S_t[3], \\
\alpha_3 &= A(S_t[7] \oplus S_t[3]) \oplus S_t[1] \oplus S_t[6], \\
\alpha_4 &= A(A(S_t[7]) \oplus S_t[0] \oplus S_t[6]) \oplus A(S_t[3]) \oplus A(S_t[2]) \oplus S_t[1], \\
\alpha_5 &= A(S_t[0] \oplus S_t[6] \oplus A(S_t[2]) \oplus S_t[1]) \oplus A(S_t[0]) \oplus S_t[7] \oplus A(S_t[5]) \oplus S_t[4], \\
\alpha_6 &= A(A(S_t[0] \oplus S_t[6]) \oplus S_t[7] \oplus A(S_t[5]) \oplus S_t[4]) \\
&\quad \oplus A(A(S_t[2]) \oplus S_t[1]) \oplus A(S_t[1] \oplus S_t[6]) \oplus A(S_t[0]) \oplus S_t[7], \\
\alpha_7 &= A(A(S_t[5]) \oplus S_t[4] \oplus S_t[1] \oplus S_t[6] \oplus A(S_t[0])) \\
&\quad \oplus A(S_t[7]) \oplus S_t[0] \oplus S_t[6] \oplus A(A(S_t[4]) \oplus S_t[3]) \oplus S_t[3],
\end{aligned}
$$

where $\alpha_i \in \mathbb{F}_2^{128}$ $(0 \leq i \leq 7)$ are known constants. It is obvious that the attackers should not completely guess 2 state blocks as the time complexity of guess will be 2^{256}. A clever way is to guess a column and a diagonal of the state blocks, which fits well with the form of the outputs. Such a strategy will allow attackers to guess at most 8 columns and diagonals. However, only in the conditions imposed by $(\alpha_0, \alpha_1, \alpha_3)$, one AES round is involved, i.e. the clever way is only applicable to these conditions. For the remaining conditions, two AES rounds are involved, which implies that the attackers at least need to guess a complete 128-bit block due to the full diffusion. For such reasons, we believe the time complexity of the guess-and-determine attack cannot be lower than 2^{256}.

7.4.4.2 Algebraic attack. It is well-known that the S-box of AES can be expressed as a set of quadratic boolean equations if the input zero is discarded. Therefore, the above equation system can be described as quadratic boolean equations by introducing extra intermediate variables to represent the outputs of the S-box for each AES round function. Notice that for different ciphertext blocks $(\alpha_0, ..., \alpha_7)$, the attackers have to introduce different variables due to the big difference between the equations. Although the system of equations is overdefined, the number of equations is only slightly larger than the number of variables and the number of variables is much larger than 256. As far as we know, such a system of equations can not be solved with time complexity 2^{256}.

7.4.5 The Linear Bias

Exploiting the fact that the output (keystream) of AEGIS is quadratic in terms of several state blocks and only one-round update is used to process each message block, Minaud proposed a statistical attack [129] on the keystream of AEGIS-256. Such an attack was improved in [80] with an automatic search method based on [160]. Specifically, the authors first utilized a simple truncated model and evaluated the minimal number of active S-boxes. It is found that for AEGIS-128, AEGIS-128L and AEGIS-256, all the results obtained in the simple truncated model suggest they are insecure against such a statistical attack. However, when searching for compatible linear trails in the bit level, almost all of them are incompatible. In addition, the results obtained in the refined model is far larger than that obtained in the simple truncated model.

To evaluate the resistance of our construction against such a statistical attack, we also adopted the simple truncated model as in [80]. According to our results, the best case is to consider 4 consecutive rounds and the minimal number of active S-boxes is 38, which suggests that the time complexity of the distinguishing attack is at least 2^{228}. Achieving 42 active S-boxes is ambitious without affecting the performance and we believe 38 is enough to resist against such an attack considering the big gap between the truncated model and bitwise model as reported in [80]. To

further verify whether there is a compatible linear trail to the best solution obtained with the truncated model, we also implemented the bitwise model where there is no additional constraint on the input mask and output mask of the S-box except the trivial infeasible pairs caused by the zero input mask or zero output mask. When searching for a compatible linear trail based on the truncated pattern, it is soon shown to be infeasible. One main reason is that compared with the attack on AEGIS-256 requiring 2 consecutive rounds, this statistical attack on Rocca requires 4 consecutive rounds, which makes the contradictions in the solutions obtained with the simple truncated model occur more easily if verified with the bitwise model. Taking this fact into account, we further believe Rocca is secure against this attack vector.

7.4.6 The State-recovery Attack Using the Decryption Oracle

In a recent work [95], by using a trivial decryption oracle, it is possible to recover the full internal state after the initialization phase with time complexity 2^{128}. Indeed, such a state-recovery attack has been observed by the designers of AEGIS-256 and it is inavoidable if the tag size is small. However, what we need to care is to prevent the further key-recovery attacks after the internal state is recovered in such a way. In AEGIS-256, this is ensured by using a keyed permutation for the initialization phase. In this revised version, we simply use a key feed-foward operation to prevent the further key-recobery attack because the attackers cannot invert the initialization phase without knowing the key even if the state after this phase is fully known. Moreover, although the finalization phase of AEGIS-256 does not involve the key, i.e. it is a public permutation, as suggested by [95], we feel it is reasonable to involve the key addition at the very beginning of this phase to prevent further forgery attacks.

7.4.7 Other Attacks

While there are many attack vectors for block ciphers, their application to Rocca is restrictive as the attackers can only know partial information of the internal state from the ciphertext blocks. In other words, reversing the round update function is impossible in Rocca without guessing many secret state blocks. For this reason, only the above potential attacks vectors are taken into account. In addition, due to the usage of the constant (Z_0, Z_1) at the initialization phase, the attack based on the similarity in the four columns of the AES state is also excluded.

7.4.8 No Claims

We do not claim the security of our scheme in the nonce-misuse setting and it seems trivial to achieve the state recovery in this setting as the output is computed with only one-round update function at the encryption phase. In addition, we do not claim the security of our scheme in the related-key and known-key setting, which is far from meaningful in real-world applications. For the attacks on the initialization phase, we emphasize that the attackers can only derive information from the restricted outputs and cannot know the full secret internal state.

7.5 Software Implementation

According to [98], target peak data rates for 5G communication are 10 Gbps for uplink and 20 Gbps for downlink. SNOW-V [81] is a new version of SNOW-family designed for 5G communication with 256-bit key support and achieves 58.25 Gbps on Intel(R) Core(TM) i7 8650U CPU @1.90GHz in encryption only mode. In the next generation (*i.e.* 6G), the target peak data rate is further increased to 100 Gbps to 1 Tbps [117]. In order to realize this high peak data rate, a new encryption algorithm is required.

Table 7.8: Performance Evaluation

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	64.60 Gbps	63.43 Gbps	57.53 Gbps	43.44 Gbps	28.94 Gbps
AEGIS-128L		104.91 Gbps	102.71 Gbps	66.28 Gbps	31.30 Gbps	14.10 Gbps
Tiaoxin-346 v2		127.55 Gbps	126.73 Gbps	81.27 Gbps	33.78 Gbps	13.61 Gbps
AEGIS-256	256-bit	66.02 Gbps	64.39 Gbps	59.09 Gbps	40.59 Gbps	26.28 Gbps
AES-256-CBC		9.35 Gbps	9.34 Gbps	9.51 Gbps	9.23 Gbps	9.26 Gbps
AES-256-CTR		58.19 Gbps	56.83 Gbps	48.77 Gbps	38.90 Gbps	19.54 Gbps
ChaCha20		11.49 Gbps	11.38 Gbps	11.40 Gbps	10.63 Gbps	4.8 Gbps
SNOW-V		43.39 Gbps	41.47 Gbps	41.59 Gbps	36.29 Gbps	25.78 Gbps
Rocca		**180.55 Gbps**	**177.71 Gbps**	**151.22 Gbps**	**98.30 Gbps**	**33.74 Gbps**
AEAD						
AEGIS-128	128-bit	60.03 Gbps	55.16 Gbps	30.13 Gbps	11.88 Gbps	3.62 Gbps
AEGIS-128L		97.55 Gbps	85.41 Gbps	31.14 Gbps	9.96 Gbps	2.95 Gbps
Tiaoxin-346 v2		114.61 Gbps	97.52 Gbps	31.67 Gbps	9.16 Gbps	2.54 Gbps
AEGIS-256	256-bit	61.16 Gbps	57.51 Gbps	30.43 Gbps	11.26 Gbps	3.37 Gbps
AES-256-GCM		29.08 Gbps	27.90 Gbps	18.78 Gbps	8.41 Gbps	2.57 Gbps
ChaCha20-Poly1305		7.60 Gbps	7.32 Gbps	5.98 Gbps	3.61 Gbps	1.24 Gbps
SNOW-V-GCM		30.20 Gbps	29.31 Gbps	19.14 Gbps	8.84 Gbps	2.73 Gbps
Rocca		**150.95 Gbps**	**131.41 Gbps**	**42.39 Gbps**	**12.75 Gbps**	**3.29 Gbps**

We evaluate the performance of Rocca and show that Rocca can achieve 160 Gbps when encrypting data of large size. Modern CPUs are equipped with a dedicated instructions set for AES called AES New Instructions (AES-NI). As Rocca has the AES round function as its component, we can optimize the implementation by utilizing AES-NI. Specifically, we use `_mm_aesenc_si128()` for AES's round function. For XORing two 128-bit values, we use `_mm_xor_si128()`. We also compare the performance with existing algorithms and demonstrate Rocca's advantage in terms of the performance. All evaluations were performed on a PC with Intel(R) Core(TM) i7-1068NG7 CPU @ 2.30GHz with 32GB RAM. For the fair comparison, we included Rocca as well as SNOW-V, Tiaoxin and AEGIS to Openssl (3.1.0-dev) and measured their performances. We used SNOW-V reference implementation with SIMD, which was given in [81]. For Tiaoxin-346 and AEGIS, we used implementations available at `https://github.com/floodyberry/supercop`. The results are given in Table 7.8, and all performance results are given in Gbps. In TLS, data will be divided into chunks of $2^{14} = 16384$ bytes or less before it is encrypted, the values in Table 7.8 are close to what we expect in practice. As shown, Rocca is 4.16 times faster than SNOW-V, and 3.10 times faster than AES-256-CTR in processing 16384 bytes message. It also outperforms both 128-bit algorithms which we tested. In encryption only mode, the initialization is performed once and only the encryption is iterated. While in AEAD mode, the initialization, associated data addition, encryption, tag generation and finalization are iterated. Here, the size of associated data is fixed to 13 bytes. In case of Rocca, the round function is iterated 20 times in the initialization and finalization, respectively, which is equivalent to processing 1280 bytes of input. As a result, we expect $1280/16384 \approx 8\%$ overhead to the encryption mode for 16384 bytes input. Additional overhead will be incurred by calling functions for the initialization, tag generation and finalization. The performance results on other CPUs are given in Sect. 7.5.1, and Rocca achieves the best performance in other CPUs as well.

The performance can be further improved by using new instructions set and/or optimizing the implementation. The new instructions set AVX512 contains `_mm512_aesenc_epi128()`, which runs four 128-bit AES round functions in parallel. As Rocca uses four AES round functions in one state update, using `_mm512_aesenc_epi128()` instead of four `_mm_aesenc_epi128()`s can be improved by up-to 4 times.

7.5.1 Software Implementation Results on Other CPUs

We show software implementation results on other CPUs in Tables 7.9 to 7.11. The evaluations were performed on Windows 10 Pro 21H1 for Table 7.9, Windows 10 Pro 21H2 for Table 7.10 and macOS Big Sur 11.4 for Tables 7.11. The difference of the environments affects the performance

of some algorithms(*e.g.* AESGIS-256, AES-256-CTR and ChaCha20), Rocca shows competitive performance on all environments.

Table 7.9: Performance on Intel(R) Core(TM) i9-12900K CPU with 64 GB RAMs.

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	101.50 Gbps	99.13 Gbps	84.44 Gbps	63.66 Gbps	23.46 Gbps
AEGIS-128L		143.87 Gbps	142.70 Gbps	126.06 Gbps	77.85 Gbps	20.08 Gbps
Tiaoxin-346 v2		192.64 Gbps	189.01 Gbps	148.12 Gbps	78.02 Gbps	21.05 Gbps
AEGIS-256	256-bit	47.27 Gbps	46.65 Gbps	45.82 Gbps	43.09 Gbps	26.55 Gbps
AES-256-CBC		13.62 Gbps	13.69 Gbps	13.65 Gbps	13.68 Gbps	13.44 Gbps
AES-256-CTR		77.82 Gbps	77.49 Gbps	68.58 Gbps	51.40 Gbps	22.04 Gbps
ChaCha20		32.98 Gbps	32.99 Gbps	31.19 Gbps	15.58 Gbps	7.67 Gbps
SNOW-V		62.00 Gbps	62.06 Gbps	56.88 Gbps	54.66 Gbps	27.09 Gbps
Rocca		**235.45 Gbps**	**232.81 Gbps**	**218.54 Gbps**	**160.92 Gbps**	**54.65 Gbps**
AEAD						
AEGIS-128	128-bit	92.66 Gbps	84.70 Gbps	38.52 Gbps	13.77 Gbps	3.68 Gbps
AEGIS-128L		125.42 Gbps	110.38 Gbps	41.49 Gbps	12.74 Gbps	3.21 Gbps
Tiaoxin-346 v2		163.23 Gbps	138.51 Gbps	46.23 Gbps	13.65 Gbps	3.55 Gbps
AEGIS-256	256-bit	44.82 Gbps	42.46 Gbps	27.98 Gbps	12.53 Gbps	3.64 Gbps
AES-256-GCM		57.87 Gbps	54.47 Gbps	29.12 Gbps	11.45 Gbps	3.13 Gbps
ChaCha20-Poly1305		21.99 Gbps	21.33 Gbps	13.99 Gbps	5.38 Gbps	1.81 Gbps
SNOW-V-GCM		36.10 Gbps	34.81 Gbps	23.63 Gbps	11.26 Gbps	3.54 Gbps
Rocca		**210.67 Gbps**	**185.90 Gbps**	**70.40 Gbps**	**22.59 Gbps**	**6.07 Gbps**

Table 7.10: Performance on Intel(R) Core(TM) i9-11900 CPU@2.50GHz with 64 GB RAMs.

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	97.67 Gbps	95.76 Gbps	79.37 Gbps	53.05 Gbps	23.53 Gbps
AEGIS-128L		142.44 Gbps	140.26 Gbps	112.17 Gbps	62.09 Gbps	17.07 Gbps
Tiaoxin-346 v2		178.69 Gbps	173.88 Gbps	120.69 Gbps	60.13 Gbps	18.79 Gbps
AEGIS-256	256-bit	34.90 Gbps	34.53 Gbps	33.09 Gbps	28.61 Gbps	18.83 Gbps
AES-256-CBC		13.71 Gbps	13.71 Gbps	13.68 Gbps	13.59 Gbps	13.26 Gbps
AES-256-CTR		84.30 Gbps	83.28 Gbps	71.92 Gbps	48.46 Gbps	21.28 Gbps
ChaCha20		62.07 Gbps	61.41 Gbps	57.25 Gbps	29.95 Gbps	7.96 Gbps
SNOW-V		63.67 Gbps	63.44 Gbps	57.21 Gbps	48.58 Gbps	25.76 Gbps
Rocca		**207.07 Gbps**	**203.49 Gbps**	**156.64 Gbps**	**80.02 Gbps**	**24.61 Gbps**
AEAD						
AEGIS-128	128-bit	88.96 Gbps	81.84 Gbps	37.80 Gbps	13.69 Gbps	3.82 Gbps
AEGIS-128L		122.92 Gbps	105.54 Gbps	36.04 Gbps	11.40 Gbps	2.99 Gbps
Tiaoxin-346 v2		148.90 Gbps	125.89 Gbps	38.13 Gbps	11.28 Gbps	3.01 Gbps
AEGIS-256	256-bit	33.86 Gbps	32.56 Gbps	22.45 Gbps	10.82 Gbps	3.49 Gbps
AES-256-GCM		117.60 Gbps	103.47 Gbps	38.44 Gbps	15.57 Gbps	4.51 Gbps
ChaCha20-Poly1305		39.26 Gbps	37.36 Gbps	17.51 Gbps	5.76 Gbps	1.60 Gbps
SNOW-V-GCM		35.43 Gbps	33.91 Gbps	21.54 Gbps	9.51 Gbps	2.93 Gbps
Rocca		**175.12 Gbps**	**148.95 Gbps**	**46.22 Gbps**	**13.81 Gbps**	**3.68 Gbps**

Table 7.11: Performance on Intel(R) Core(TM) i9-10910 CPU@3.60GHz with 64 GB RAMs.

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	73.43 Gbps	73.84 Gbps	71.21 Gbps	66.15 Gbps	29.23 Gbps
AEGIS-128L		137.50 Gbps	138.47 Gbps	97.32 Gbps	50.41 Gbps	17.45 Gbps
Tiaoxin-346 v2		163.10 Gbps	159.46 Gbps	107.99 Gbps	47.57 Gbps	14.61 Gbps
AEGIS-256	256-bit	89.14 Gbps	88.71 Gbps	82.87 Gbps	67.46 Gbps	29.06 Gbps
AES-256-CBC		10.01 Gbps	10.01 Gbps	9.98 Gbps	9.79 Gbps	9.51 Gbps
AES-256-CTR		41.39 Gbps	40.88 Gbps	39.25 Gbps	33.84 Gbps	20.15 Gbps
ChaCha20		15.89 Gbps	15.85 Gbps	15.35 Gbps	14.42 Gbps	6.86 Gbps
SNOW-V		55.44 Gbps	57.58 Gbps	55.54 Gbps	49.77 Gbps	33.90 Gbps
Rocca		**186.22 Gbps**	**186.02 Gbps**	**159.62 Gbps**	**89.59 Gbps**	**29.53 Gbps**
AEAD						
AEGIS-128	128-bit	70.04 Gbps	66.94 Gbps	38.53 Gbps	15.33 Gbps	4.18 Gbps
AEGIS-128L		122.96 Gbps	108.75 Gbps	41.34 Gbps	13.24 Gbps	3.45 Gbps
Tiaoxin-346 v2		136.70 Gbps	117.28 Gbps	33.48 Gbps	10.67 Gbps	2.94 Gbps
AEGIS-256	256-bit	79.60 Gbps	76.58 Gbps	38.85 Gbps	14.06 Gbps	3.91 Gbps
AES-256-GCM		26.32 Gbps	25.70 Gbps	17.94 Gbps	8.79 Gbps	2.85 Gbps
ChaCha20-Poly1305		9.95 Gbps	10.02 Gbps	8.00 Gbps	4.69 Gbps	1.59 Gbps
SNOW-V-GCM		31.56 Gbps	30.55 Gbps	19.66 Gbps	8.97 Gbps	2.83 Gbps
Rocca		**156.15 Gbps**	**132.09 Gbps**	**41.99 Gbps**	**12.60 Gbps**	**3.30 Gbps**

We also evaluate the performance on Android and iOS, implemented with ARM NEON intrinsics. The results are shown in the Tables 7.12 to 7.14. Note that the implementation of SNOW-V is not optimized and the shown results can be further improved by optimizing the implementation. In

the original paper, Ekdahl *et al.* [81] showed SNOW-V can achieve 23.59 Gbps on Apple A11 SoC. Rocca achieves very competitive performance on recent mobile platforms. The performance is improved on the newer platforms (*i.e.* Snapdragon 888 and A15 Bionic) and further improvement is expected in the future.

Table 7.12: Performance on Apple M1

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	41.24 Gbps	41.20 Gbps	40.50 Gbps	36.81 Gbps	26.73 Gbps
AEGIS-128L		67.58 Gbps	67.19 Gbps	61.01 Gbps	46.42 Gbps	23.39 Gbps
Tiaoxin-346 v2		90.45 Gbps	89.23 Gbps	75.47 Gbps	49.47 Gbps	21.33 Gbps
AEGIS-256	256-bit	39.95 Gbps	39.77 Gbps	39.00 Gbps	36.12 Gbps	26.29 Gbps
AES-256-CBC		8.44 Gbps	8.46 Gbps	8.40 Gbps	8.30 Gbps	7.91 Gbps
AES-256-CTR		69.11 Gbps	68.94 Gbps	63.77 Gbps	42.12 Gbps	26.79 Gbps
ChaCha20		18.77 Gbps	18.75 Gbps	18.68 Gbps	10.43 Gbps	5.10 Gbps
SNOW-V		29.43 Gbps	29.52 Gbps	29.15 Gbps	28.28 Gbps	25.22 Gbps
Rocca		**96.54 Gbps**	**96.58 Gbps**	**91.03 Gbps**	**71.48 Gbps**	**35.66 Gbps**
AEAD						
AEGIS-128	128-bit	40.22 Gbps	39.21 Gbps	28.98 Gbps	15.14 Gbps	5.18 Gbps
AEGIS-128L		64.35 Gbps	61.33 Gbps	35.69 Gbps	14.55 Gbps	4.30 Gbps
Tiaoxin-346 v2		85.43 Gbps	78.75 Gbps	37.26 Gbps	13.38 Gbps	3.74 Gbps
AEGIS-256	256-bit	38.68 Gbps	37.7 Gbps	26.53 Gbps	13.23 Gbps	4.45 Gbps
AES-256-GCM		41.71 Gbps	40.18 Gbps	27.37 Gbps	15.18 Gbps	5.44 Gbps
ChaCha20-Poly1305		13.36 Gbps	13.12 Gbps	9.77 Gbps	4.41 Gbps	1.73 Gbps
SNOW-V-GCM		8.01 Gbps	7.92 Gbps	6.82 Gbps	4.59 Gbps	1.99 Gbps
Rocca		**90.02 Gbps**	**82.12 Gbps**	**37.30 Gbps**	**12.94 Gbps**	**3.56 Gbps**

Table 7.13: Performance on Apple A15 Bionic

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	43.28 Gbps	43.34 Gbps	42.04 Gbps	36.50 Gbps	23.33 Gbps
AEGIS-128L		69.25 Gbps	68.09 Gbps	60.87 Gbps	44.61 Gbps	20.18 Gbps
Tiaoxin-346 v2		87.48 Gbps	86.26 Gbps	71.58 Gbps	43.91 Gbps	17.34 Gbps
AEGIS-256	256-bit	39.21 Gbps	39.35 Gbps	38.21 Gbps	34.76 Gbps	22.29 Gbps
AES-256-CBC		8.74 Gbps	8.77 Gbps	9.07 Gbps	8.99 Gbps	8.40 Gbps
AES-256-CTR		69.39 Gbps	69.06 Gbps	62.95 Gbps	46.76 Gbps	25.34 Gbps
ChaCha20		17.21 Gbps	17.04 Gbps	16.68 Gbps	10.00 Gbps	5.07 Gbps
SNOW-V		29.75 Gbps	29.78 Gbps	28.92 Gbps	27.24 Gbps	22.46 Gbps
Rocca		**103.09 Gbps**	**102.45 Gbps**	**91.23 Gbps**	**70.86 Gbps**	**32.85 Gbps**
AEAD						
AEGIS-128	128-bit	40.61 Gbps	39.27 Gbps	27.39 Gbps	13.25 Gbps	4.39 Gbps
AEGIS-128L		63.10 Gbps	59.38 Gbps	32.32 Gbps	12.58 Gbps	3.68 Gbps
Tiaoxin-346 v2		81.25 Gbps	74.24 Gbps	33.29 Gbps	11.51 Gbps	3.20 Gbps
AEGIS-256	256-bit	39.00 Gbps	37.54 Gbps	25.27 Gbps	11.98 Gbps	3.74 Gbps
AES-256-GCM		43.87 Gbps	42.42 Gbps	28.26 Gbps	14.30 Gbps	4.82 Gbps
ChaCha20-Poly1305		12.30 Gbps	12.07 Gbps	9.18 Gbps	4.45 Gbps	1.76 Gbps
SNOW-V-GCM		7.73 Gbps	7.66 Gbps	6.47 Gbps	4.14 Gbps	1.73 Gbps
Rocca		**82.20 Gbps**	**76.59 Gbps**	**33.40 Gbps**	**11.28 Gbps**	**3.05 Gbps**

Table 7.14: Performance on Qualcomm Snapdragon 888

Algorithms	Key length	Size of input (bytes)				
		16384	8192	1024	256	64
Encryption only						
AEGIS-128	128-bit	31.33 Gbps	30.92 Gbps	30.03 Gbps	27.44 Gbps	19.96 Gbps
AEGIS-128L		53.35 Gbps	53.06 Gbps	47.44 Gbps	34.37 Gbps	16.61 Gbps
Tiaoxin-346 v2		63.41 Gbps	62.57 Gbps	52.50 Gbps	34.25 Gbps	13.84 Gbps
AEGIS-256	256-bit	34.26 Gbps	34.38 Gbps	32.75 Gbps	28.54 Gbps	19.54 Gbps
AES-256-CBC		11.76 Gbps	11.82 Gbps	11.65 Gbps	11.46 Gbps	10.61 Gbps
AES-256-CTR		37.70 Gbps	37.45 Gbps	35.18 Gbps	27.44 Gbps	17.37 Gbps
ChaCha20		13.44 Gbps	13.40 Gbps	13.04 Gbps	8.22 Gbps	4.23 Gbps
SNOW-V		23.94 Gbps	23.85 Gbps	23.59 Gbps	22.12 Gbps	18.44 Gbps
Rocca		**78.11 Gbps**	**77.51 Gbps**	**70.08 Gbps**	**51.55 Gbps**	**24.07 Gbps**
AEAD						
AEGIS-128	128-bit	30.63 Gbps	30.24 Gbps	20.68 Gbps	9.98 Gbps	3.26 Gbps
AEGIS-128L		50.63 Gbps	47.35 Gbps	24.34 Gbps	9.16 Gbps	2.65 Gbps
Tiaoxin-346 v2		58.45 Gbps	53.39 Gbps	24.26 Gbps	8.35 Gbps	1.20 Gbps
AEGIS-256	256-bit	32.74 Gbps	31.37 Gbps	20.00 Gbps	8.97 Gbps	2.76 Gbps
AES-256-GCM		25.44 Gbps	24.38 Gbps	17.02 Gbps	8.29 Gbps	2.88 Gbps
ChaCha20-Poly1305		9.21 Gbps	8.94 Gbps	6.29 Gbps	3.13 Gbps	1.21 Gbps
SNOW-V-GCM		7.25 Gbps	7.15 Gbps	5.78 Gbps	3.50 Gbps	1.36 Gbps
Rocca		**71.23 Gbps**	**64.00 Gbps**	**26.55 Gbps**	**8.80 Gbps**	**2.38 Gbps**

7.6 Conclusions

To fulfill the basic requirements on the speed and security in 6G systems, i.e. 100 Gbps and 256-bit security, we are motivated to further study the generalized method to construct round functions based on the parallel calls to the AES round function, which was first studied by Jean and Nikolić in FSE 2016. As a result, an efficient AES-based AEAD scheme called Rocca is proposed, whose construction is only based on the AES round function and the 128-bit XOR operation supported by the SIMD instructions on model CPUs. In addition, we have performed a thorough study to understand the security of Rocca. According to the software implementation, Rocca can reach 150 Gbps in the AEAD mode, which is more than four times faster than SNOW-V designed for 5G systems. To the best of our knowledge, Rocca is the first dedicated scheme targeting 6G systems and it also shows the potential to reach the basic requirements in such systems.

As future work, a parallelizable mode of Rocca would be interesting and beneficial for both environments equipped with multiple cores and not supported AES-NI.

8 Conclusions

In this thesis, we studied on how to design efficient symmetric-key cryptographic algorithms. More specifically, we introduce four latest algorithms, including a lightweight block cipher, lightweight tweakable block cipher, lightweight block cipher-based PRF, and ultra-high throughput AEAD as follows:

We introduced a lightweight tweakable block cipher Tweakable TWINE whose the key, block, and tweak sizes are 80/128, 64, and 64 bits, respectively. By investigating the best linear layer for the tweak scheduling function, we expanded a block cipher TWINE to a tweakable block cipher Tweakable TWINE with a minimum additional hardware cost. As a result, Tweakable TWINE achieves a competitive hardware implementation result compared with other tweakable block ciphers.

We introduced a lightweight block cipher WARP whose the key and block sizes are both 128 bits. By carefully revisiting all components of type-2 GFN, we could construct a 128-bit block cipher with a small GEs from scratch. As a result, WARP achieves the minimum hardware circuit size among 128-bit block ciphers.

We introduced a block cipher-based low latency PRF Orthros whose the key and block sizes are both 128 bits. With the two branches-based construction, we could design ultra low-latency PRF due to no need to consider of the additional rounds to stand the key-recovery attack. As a result, Orthros acieves the minimum latency among all block ciphers.

We introduced an ultra-high throughput AEAD scheme Rocca for beyond 5G applications. Rocca is optimized on a pure software environment by taking full advantage of the SIMD instruction and AES-NI, both of which are equipped with the latest CPUs and SoCs. As a result, Rocca achieves incredible encryption/decryption speed of more than 200 Gbps on the latest CPU. Moreover, Rocca also supports 258-bit secret key to resist against future attacks with a quantum computer. We emphasize that Rocca is the first ever encryption scheme meeting both security and performance requirements for beyond 5G systems.

Acknowledgements

First of all, I would like to express my gratitude to my supervisor Takanori Isobe for his all support and advice to my research. He introduced me to research in the field of symmetric-key cryptography and has also given me many valuable opportunities to start my career as a researcher. There is no doubt that this five-year experience, including my master's and PhD studies, is going to help me with shaping my future career. Without him, this work would not have been possible, and I could not have started my career as a cryptographic researcher.

I am really grateful for Kazuhiko Minematsu in NEC corporation. He has given me a lot of advice to my research since when the beginning of my research career in the field of symmetric-key cryptography. Despite being a mere master's student with limited knowledge in the field, he graciously accepted me for an internship and it was through this experience that I developed a passion for research. I may not have pursued a PhD without him

I would like to thank Fukang Liu. He has helped me a lot with my research since when I was a master student. Without his help, I would not have finished this work. In addition, I would like to express my appreciation to all of my co-authors and lab members who have supported me throughout my master's and PhD studies. I really appreciate Prof. Yukikazu Nakamoto and Prof. Jun Kurihara for judging and reviewing this thesis.

Finally, I greatly appreciate my parent, siblings, and grandparents for their long-time support.

Kosei Sakamoto
Kobe, January 2023

References

[1] *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC.* National Institute of Standards and Technology. Springer, 2007.

[2] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness,. `https://competitions.cr.yp.to/caesar.html`, 2018.

[3] 3GPP SA3. Study on the support of 256-bit algorithms for 5G. `https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3422`, 2018.

[4] Ahmed Abdelkhalek, Yu Sasaki, Yosuke Todo, Mohamed Tolba, and Amr M. Youssef. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Trans. Symmetric Cryptol.*, 2017(4):99–129, 2017.

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

[6] Elena Andreeva, Andrey Bogdanov, Nilanjan Datta, Atul Luykx, Bart Mennink, Mridul Nandi, Elmar Tischhauser, and Kan Yasuda. Colm v1. a CAESAR portfolio, 2016.

[7] arm. Arm® architecture reference manual armv8, for armv8-a architecture profile, 2021.

[8] Roberto Avanzi. The QARMA block cipher family – almost MDS matrices over rings with zero divisors, nearly symmetric Even-Mansour constructions with non-involutory central rounds, and search heuristics for low-latency S-boxes. Cryptology ePrint Archive, Report 2016/444, 2016. `http://eprint.iacr.org/2016/444`.

[9] Roberto Avanzi. The QARMA block cipher family. *IACR Trans. Symm. Cryptol.*, 2017(1):4–44, 2017.

[10] Subhadeep Banik, Zhenzhen Bao, Takanori Isobe, Hiroyasu Kubo, Fukang Liu, Kazuhiko Minematsu, Kosei Sakamoto, Nao Shibata, and Maki Shigeri. WARP : Revisiting GFN for lightweight 128-bit block cipher. In *SAC*, volume 12804 of *Lecture Notes in Computer Science*, pages 535–564. Springer, 2020.

[11] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 411–436. Springer, Heidelberg, November / December 2015.

[12] Subhadeep Banik, Andrey Bogdanov, Atul Luykx, and Elmar Tischhauser. SUNDAE: Small universal deterministic authenticated encryption for the internet of things. *IACR Trans. Symm. Cryptol.*, 2018(3):1–35, 2018.

[13] Subhadeep Banik, Andrey Bogdanov, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, Elmar Tischhauser, and Yosuke Todo. Sundae-gift. A Submission to NIST Lightweight Cryptography Project, 2019.

[14] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Exploring energy efficiency of lightweight block ciphers. In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 178–194, 2015.

[15] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-AES: A compact implementation of the AES encryption/decryption core. In Orr Dunkelman and Somitra Kumar Sanadhya, editors, *INDOCRYPT 2016*, volume 10095 of *LNCS*, pages 173–190. Springer, Heidelberg, December 2016.

[16] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Atomic-aes: A compact implementation of the AES encryption/decryption core. In *Progress in Cryptology - INDOCRYPT 2016 - 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings*, pages 173–190, 2016.

[17] Subhadeep Banik, Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, Mridul Nandi, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. Gift-cofb. A Submission to NIST Lightweight Cryptography Project, 2019.

[18] Subhadeep Banik, Takanori Isobe, Fukang Liu, Kazuhiko Minematsu, and Kosei Sakamoto. Orthros: A low-latency PRF. *IACR Trans. Symmetric Cryptol.*, 2021(1):37–77, 2021.

[19] Subhadeep Banik, Vasily Mikhalev, Frederik Armknecht, Takanori Isobe, Willi Meier, Andrey Bogdanov, Yuhei Watanabe, and Francesco Regazzoni. Towards low energy stream ciphers. *IACR Trans. Symm. Cryptol.*, 2018(2):1–19, 2018.

[20] 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 Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 321–345. Springer, Heidelberg, September 2017.

[21] Zhenzhen Bao, Chun Guo, Jian Guo, and Ling Song. TNT: How to tweak a block cipher. In Vincent Rijmen and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, LNCS, pages 641–673. Springer, Heidelberg, May 2020.

[22] Achiya Bar-On, Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. Improved key recovery attacks on reduced-round AES with practical data and memory complexities. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 185–212. Springer, Heidelberg, August 2018.

[23] Navid Ghaedi Bardeh and Sondre Rønjom. The exchange attack: How to distinguish six rounds of AES with $2^{88.2}$ chosen plaintexts. LNCS, pages 347–370. Springer, Heidelberg, December 2019.

[24] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. `http://eprint.iacr.org/2013/404`.

[25] Christof Beierle, Anne Canteaut, Gregor Leander, and Yann Rotella. Proving resistance against invariant attacks: How to choose the round constants. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 647–678, 2017.

[26] 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 (2)*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[27] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 123–153. Springer, Heidelberg, August 2016.

[28] 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. Cryptology ePrint Archive, Report 2016/660, 2016. `http://eprint.iacr.org/2016/660`.

[29] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Trans. Symmetric Cryptol.*, 2019(1):5–45, 2019.

[30] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: Lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Trans. Symm. Cryptol.*, 2019(1):5–45, 2019.

[31] M. Bellare and R. Impagliazzo. A tool for obtaining tighter security analyses of pseudorandom function based constructions, with applications to PRP to PRF conversion. Cryptology ePrint Archive, Report 1999/024, 1999. `http://eprint.iacr.org/1999/024`.

[32] Ryad Benadjila, Jian Guo, Victor Lomné, and Thomas Peyrin. Implementing lightweight block ciphers on x86 architectures. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 324–351. Springer, Heidelberg, August 2014.

[33] Thierry P. Berger, Julien Francq, Marine Minier, and Gaël Thomas. Extended generalized feistel networks using matrix representation to propose a new lightweight block cipher: Lilliput. *IEEE Trans. Computers*, 65(7):2074–2089, 2016.

[34] Thierry P. Berger, Marine Minier, and Gaël Thomas. Extended generalized Feistel networks using matrix representation. In Tanja Lange, Kristin Lauter, and Petr Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 289–305. Springer, Heidelberg, August 2014.

[35] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Team keccak: Hardware resources. `https://keccak.team/hardware.html`.

[36] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

[37] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography*, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[38] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, Ronny Van Keer, and Benoît Viguier. Kangarootwelve: Fast hashing based on keccak-p. In *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, pages 400–418, 2018.

[39] Karthikeyan Bhargavan and Gaëtan Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 456–467. ACM Press, October 2016.

[40] Eli Biham, Alex Biryukov, Orr Dunkelman, Eran Richardson, and Adi Shamir. Initial observations on Skipjack: Cryptanalysis of Skipjack-3XOR (invited talk). In Stafford E. Tavares and Henk Meijer, editors, *SAC 1998*, volume 1556 of *LNCS*, pages 362–376. Springer, Heidelberg, August 1999.

[41] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[42] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 12–23. Springer, Heidelberg, May 1999.

[43] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1990.

[44] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 2–21. Springer, Heidelberg, August 1991.

[45] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 487–496. Springer, Heidelberg, August 1993.

[46] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. Threshold implementations of all 3 ×3 and 4 ×4 s-boxes. In *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, pages 76–91, 2012.

[47] Alex Biryukov. The design of a stream cipher LEX. In Eli Biham and Amr M. Youssef, editors, *SAC 2006*, volume 4356 of *LNCS*, pages 67–75. Springer, Heidelberg, August 2007.

[48] Alex Biryukov and Christophe De Cannière. Data encryption standard (DES). In *Encyclopedia of Cryptography and Security*. Springer, 2005.

[49] Alex Biryukov, Patrick Derbez, and Léo Perrin. Differential analysis and meet-in-the-middle attack against round-reduced TWINE. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 3–27. Springer, Heidelberg, March 2015.

[50] Alex Biryukov and Ivica Nikolic. Complementing Feistel ciphers. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 3–18. Springer, Heidelberg, March 2014.

[51] 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*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[52] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, Heidelberg, September 2007.

[53] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knežević, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 208–225. Springer, Heidelberg, December 2012.

[54] Christina Boura, María Naya-Plasencia, and Valentin Suder. Scrutinizing and improving impossible differential attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 179–199. Springer, Heidelberg, December 2014.

[55] Victor Cauchois, Clément Gomez, and Gaël Thomas. General diffusion analysis: How to find optimal permutations for generalized type-II Feistel schemes. *IACR Trans. Symm. Cryptol.*, 2019(1):264–301, 2019.

[56] Florent Chabaud and Serge Vaudenay. Links between differential and linear cryptanalysis. In *EUROCRYPT*, volume 950 of *Lecture Notes in Computer Science*, pages 356–365. Springer, 1994.

[57] Avik Chakraborti, Nilanjan Datta, Ashwin Jha, Cuauhtemoc Mancillas-López, Mridul Nandi, and Yu Sasaki. Elastic-tweak: A framework for short tweak tweakable block cipher. In *INDOCRYPT*, volume 13143 of *Lecture Notes in Computer Science*, pages 114–137. Springer, 2021.

[58] Avik Chakraborti, Tetsu Iwata, Kazuhiko Minematsu, and Mridul Nandi. Blockcipher-based authenticated encryption: How small can we go? In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 277–298. Springer, Heidelberg, September 2017.

[59] Yu Long Chen, Eran Lambooij, and Bart Mennink. How to build pseudorandom functions from public random permutations. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 266–293. Springer, Heidelberg, August 2019.

[60] Intel Corporation. Intel Advanced Encryption Standard (AES) New Instructions Set. Official webpage, `https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf`.

[61] Intel Corporation. Intel intrinsics guide. Official webpage, https://software.intel.com/sites/landingpage/IntrinsicsGuide/.

[62] Joan Daemen. Limitations of the even-mansour construction. In *ASIACRYPT*, volume 739 of *Lecture Notes in Computer Science*, pages 495–498. Springer, 1991.

[63] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher Square. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 149–165. Springer, Heidelberg, January 1997.

[64] Joan Daemen, Pedro Maat Costa Massolino, Alireza Mehrdad, and Yann Rotella. The subterranean 2.0 cipher suite. `https://cs.ru.nl/~joan/subterranean.html`.

[65] Joan Daemen, Pedro Maat Costa Massolino, Alireza Mehrdad, and Yann Rotella. The subterranean 2.0 cipher suite. *IACR Trans. Symmetric Cryptol.*, 2020(S1):262–294, 2020.

[66] Joan Daemen, Michaël Peeters, Gilles Van Assche, and Vincent Rijmen. Nessie proposal: Noekeon. http://gro.noekeon.org/Noekeon-spec.pdf, 2000.

[67] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[68] Joan Daemen and Vincent Rijmen. A new MAC construction ALRED and a specific instance ALPHA-MAC. In Henri Gilbert and Helena Handschuh, editors, *FSE 2005*, volume 3557 of *LNCS*, pages 1–17. Springer, Heidelberg, February 2005.

[69] Joan Daemen and Vincent Rijmen. The Pelican MAC function 2.0. Cryptology ePrint Archive, Report 2005/088, 2005. `http://eprint.iacr.org/2005/088`.

[70] Wei Dai, Viet Tung Hoang, and Stefano Tessaro. Information-theoretic indistinguishability via the chi-squared method. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 497–523. Springer, Heidelberg, August 2017.

[71] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, Heidelberg, September 2009.

[72] Patrick Derbez. Note on impossible differential attacks. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 416–427. Springer, Heidelberg, March 2016.

[73] Patrick Derbez, Pierre-Alain Fouque, Baptiste Lambin, and Victor Mollimard. Efficient search for optimal diffusion layers of generalized feistel networks. 2019(1):218–240, 2019.

[74] Patrick Derbez, Tetsu Iwata, Ling Sun, Siwei Sun, Yosuke Todo, Haoyang Wang, and Meiqin Wang. Cryptanalysis of AES-PRF and its dual. *IACR Trans. Symm. Cryptol.*, 2018(2):161–191, 2018.

[75] Daniel Dinu, Yann Le Corre, Dmitry Khovratovich, Léo Perrin, Johann Großschädl, and Alex Biryukov. Triathlon of lightweight block ciphers for the internet of things. *Journal of Cryptographic Engineering*, 9(3):283–302, September 2019.

[76] Hans Dobbertin, Antoon Bosselaers, and Bart Preneel. RIPEMD-160: A strengthened version of RIPEMD. In Dieter Gollmann, editor, *FSE'96*, volume 1039 of *LNCS*, pages 71–82. Springer, Heidelberg, February 1996.

[77] Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. The retracing boomerang attack. In Vincent Rijmen and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, LNCS, pages 280–309. Springer, Heidelberg, May 2020.

[78] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. Standard, National Institute of Standards and Technology., 2005.

[79] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices. Standard, National Institute of Standards and Technology., 2010.

[80] Maria Eichlseder, Marcel Nageler, and Robert Primas. Analyzing the linear keystream biases in AEGIS. *IACR Trans. Symmetric Cryptol.*, 2019(4):348–368, 2019.

[81] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. *IACR Trans. Symmetric Cryptol.*, 2019(3):1–42, 2019.

[82] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *ASIACRYPT'91*, volume 739 of *LNCS*, pages 210–224. Springer, Heidelberg, November 1993.

[83] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl - a SHA-3 candidate. In Helena Handschuh, Stefan Lucks, Bart Preneel, and Phillip Rogaway, editors, *Symmetric Cryptography, 11.01. - 16.01.2009*, volume 09031 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 2009.

[84] David Goldenberg, Susan Hohenberger, Moses Liskov, Elizabeth Crump Schwartz, and Hakan Seyalioglu. On tweaking Luby-Rackoff blockciphers. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 342–356. Springer, Heidelberg, December 2007.

[85] Lorenzo Grassi. Mixture differential cryptanalysis: a new approach to distinguishers and attacks on round-reduced AES. *IACR Trans. Symm. Cryptol.*, 2018(2):133–160, 2018.

[86] Vincent Grosso, Gaëtan Leurent, François-Xavier Standaert, and Kerem Varici. LS-designs: Bitslice encryption for efficient masked software implementations. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 18–37. Springer, Heidelberg, March 2015.

[87] Shay Gueron. Intel advanced encryption standard (aes) new instructions set, 2010.

[88] Shay Gueron. A memory encryption engine suitable for general purpose processors. Cryptology ePrint Archive, Report 2016/204, 2016. `http://eprint.iacr.org/2016/204`.

[89] Shay Gueron and Nicky Mouha. Simpira v2: A family of efficient permutations using the AES round function. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 95–125. Springer, Heidelberg, December 2016.

[90] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 326–341. Springer, Heidelberg, September / October 2011.

[91] Kishan Chand Gupta, Sumit Kumar Pandey, and Ayineedi Venkateswarlu. Almost involutory recursive MDS diffusion layers. *Des. Codes Cryptography*, 87(2-3):609–626, 2019.

[92] W. Eric Hall and Charanjit S. Jutla. Parallelizable authentication trees. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pages 95–109. Springer, Heidelberg, August 2006.

[93] Viet Tung Hoang, Ted Krovetz, and Phillip Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 15–44. Springer, Heidelberg, April 2015.

[94] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bon-Seok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A new block cipher suitable for low-resource device. In Louis Goubin and Mitsuru Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 46–59. Springer, Heidelberg, October 2006.

[95] Akinori Hosoyamada, Akiko Inoue, Ryoma Ito, Tetsu Iwata, Kazuhiko Mimematsu, Ferdinand Sibleyras, and Yosuke Todo. Cryptanalysis of rocca and feasibility of its security claim. *IACR Transactions on Symmetric Cryptology*, 2022(3):123–151, Sep. 2022.

[96] Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices.

[97] Gurobi Optimization Inc. Gurobi optimizer 6.5. Official webpage, http://www.gurobi.com/, 2015.

[98] ITU. Minimum requirements related to technical performance for IMT-2020 radio interface(s), 2017.

[99] Tetsu Iwata. New blockcipher modes of operation with beyond the birthday bound security. In Matthew J. B. Robshaw, editor, *FSE 2006*, volume 4047 of *LNCS*, pages 310–327. Springer, Heidelberg, March 2006.

[100] Tetsu Iwata, Mustafa Khairallah, Kazuhiko Minematsu, and Thomas Peyrin. Duel of the titans: The romulus and remus families of lightweight AEAD algorithms. *IACR Trans. Symmetric Cryptol.*, 2020(1):43–120, 2020.

[101] Tetsu Iwata, Kazuhiko Minematsu, Jian Guo, and Sumio Morioka. CLOC: Authenticated encryption for short input. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 149–167. Springer, Heidelberg, March 2015.

[102] 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 *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 687–707, 2017.

[103] Jérémy Jean and Ivica Nikolic. Efficient design strategies based on the AES round function. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2016.

[104] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2014.

[105] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 274–288. Springer, Heidelberg, December 2014.

[106] Peyrin T Seurin Jean J, Nikolić I. Deoxys v1.41. submitted to CAESAR, 2016.

[107] Auguste Kerkhoff. La cryptographie militaire. ournal de sciences militaires, IX:5–38, 161–191,, 1883.

[108] Miroslav Knežević, Ventzislav Nikov, and Peter Rombouts. Low-latency encryption - is "lightweight = light + wait"? In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 426–446. Springer, Heidelberg, September 2012.

[109] Lars R. Knudsen, Gregor Leander, Axel Poschmann, and Matthew J. B. Robshaw. PRINTcipher: A block cipher for IC-printing. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 16–32. Springer, Heidelberg, August 2010.

[110] Lars R. Knudsen and David Wagner. Integral cryptanalysis. In Joan Daemen and Vincent Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 112–127. Springer, Heidelberg, February 2002.

[111] Stefan Kölbl. Avx implementation of the skinny block cipher. https://github.com/kste/skinny_avx, 2019.

[112] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In Antoine Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 306–327. Springer, Heidelberg, February 2011.

[113] Xuejia Lai, James L Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 17–38. Springer, 1991.

[114] Rodolphe Lampe and Yannick Seurin. Tweakable blockciphers with asymptotically optimal security. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 133–151. Springer, Heidelberg, March 2014.

[115] Will Landecker, Thomas Shrimpton, and R. Seth Terashima. Tweakable blockciphers with beyond birthday-bound security. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 14–30. Springer, 2012.

[116] Will Landecker, Thomas Shrimpton, and R. Seth Terashima. Tweakable blockciphers with beyond birthday-bound security. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 14–30. Springer, Heidelberg, August 2012.

[117] Matti Latva-aho and Kari Leppänen. Key drivers and research challenges for 6G ubiquitous wireless intelligence, 2019.

[118] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 31–46. Springer, Heidelberg, August 2002.

[119] Fukang Liu, Takanori Isobe, Willi Meier, and Kosei Sakamoto. Weak keys in reduced aegis and tiaoxin. Cryptology ePrint Archive, Report 2021/187, 2021. `https://eprint.iacr.org/2021/187`.

[120] Yunwen Liu, Qingju Wang, and Vincent Rijmen. Automatic search of linear trails in ARX with applications to SPECK and chaskey. In *ACNS*, volume 9696 of *Lecture Notes in Computer Science*, pages 485–499. Springer, 2016.

[121] Stefan Lucks. The sum of PRPs is a secure PRF. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 470–484. Springer, Heidelberg, May 2000.

[122] J. L. Massey. Cryptography: Fundamentals and applications. Copies of transparencies, Advanced Technology Seminars, 1993.

[123] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1993.

[124] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 386–397. Springer, Heidelberg, May 1994.

[125] Edward J McCluskey. Minimization of Boolean functions. *The Bell System Technical Journal*, 35(6):1417–1444, 1956.

[126] Bart Mennink and Samuel Neves. Encrypted Davies-Meyer and its dual: Towards optimal security using mirror theory. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 556–583. Springer, Heidelberg, August 2017.

[127] Bart Mennink and Samuel Neves. Optimal PRFs from blockcipher designs. *IACR Trans. Symm. Cryptol.*, 2017(3):228–252, 2017.

[128] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On ciphers that continuously access the non-volatile key. *IACR Trans. Symm. Cryptol.*, 2016(2):52–79, 2016. `http://tosc.iacr.org/index.php/ToSC/article/view/565`.

[129] Brice Minaud. Linear biases in AEGIS keystream. In Antoine Joux and Amr M. Youssef, editors, *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, volume 8781 of *Lecture Notes in Computer Science*, pages 290–305. Springer, 2014.

[130] Kazuhiko Minematsu. Beyond-birthday-bound security based on tweakable block cipher. In *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 308–326. Springer, 2009.

[131] Atsushi Mitsuda and Tetsu Iwata. Tweakable pseudorandom permutation from generalized Feistel structure. In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *ProvSec 2008*, volume 5324 of *LNCS*, pages 22–37. Springer, Heidelberg, October / November 2008.

[132] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 69–88. Springer, Heidelberg, May 2011.

[133] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[134] Yusuke Naito, Mitsuru Matsui, Takeshi Sugawara, and Daisuke Suzuki. SAEB: A lightweight blockcipher-based AEAD mode of operation. *IACR TCHES*, 2018(2):192–217, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/885`.

[135] National Institute of Standards and Technology. FIPS 197 Advanced encryption standard., 2001.

[136] Bruce Schneier Doug Whiting Mihir Bellare Tadayoshi Kohno Jon Callas Jesse Walker Niels Ferguson, Stefan Lucks. The skein hash function family. http://www.skein-hash.info, 2010.

[137] Ivica Nikolić. Tiaoxin-346: Version 2.0. CAESAR Competition, 2014.

[138] Y. Nir and A. Langley. *RFC 8439: ChaCha20 and Poly1305 for IETF Protocols.* IETF, 2018.

[139] Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. Technical report, 2007. National Institute of Standards and Technology.

[140] Kaisa Nyberg. Generalized Feistel networks. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT'96*, volume 1163 of *LNCS*, pages 91–104. Springer, Heidelberg, November 1996.

[141] Jacques Patarin. A proof of security in O(2n) for the xor of two random permutations. In Reihaneh Safavi-Naini, editor, *ICITS 08*, volume 5155 of *LNCS*, pages 232–248. Springer, Heidelberg, August 2008.

[142] Raphael Chung-Wei Phan. Impossible differential cryptanalysis of 7-round advanced encryption standard (AES). *Inf. Process. Lett.*, 91(1):33–38, 2004.

[143] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-channel resistant crypto for less than 2, 300 GE. *J. Cryptol.*, 24(2):322–345, 2011.

[144] Qualcomm Technologies Inc. Pointer Authentication on ARMv8.3. `https://www.qualcomm.com/media/documents/files/whitepaper-pointer-authentication-on-armv8-3.pdf`.

[145] Willard V Quine. The problem of simplifying truth functions. *The American mathematical monthly*, 59(8):521–531, 1952.

[146] Willard V Quine. A way to simplify truth functions. *The American mathematical monthly*, 62(9):627–631, 1955.

[147] Schroeppel R. An overview of the hasty pudding cipher. http://www.cs.arizona.edu/ rcs/hpc, 1998.

[148] Real-Time and Embedded Sys Lab. uops.info. Official webpage, `https://www.uops.info/`.

[149] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, Heidelberg, December 2004.

[150] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 196–205. ACM Press, November 2001.

[151] Sondre Rønjom, Navid Ghaedi Bardeh, and Tor Helleseth. Yoyo tricks with AES. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 217–243. Springer, Heidelberg, December 2017.

[152] SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2. Version 1.1, ETSI/SAGE, 2006. `https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.02.00_60/ts_133501v150200p.pdf`, 2006.

[153] SAGE. Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3. document 2: ZUC specification. Version 1.6, ETSI/SAGE, 2011. `https://www.etsi.org/deliver/etsi_ts/133500_133599/133501/15.02.00_60/ts_133501v150200p.pdf`, 2011.

[154] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient aes-based encryption scheme for beyond 5g. *IACR Trans. Symmetric Cryptol.*, 2021(2):1–30, 2021.

[155] Kosei Sakamoto, Kazuhiko Minematsu, Nao Shibata, Maki Shigeri, Hiroyasu Kubo, Yuki Funabiki, Andrey Bogdanov, Sumio Morioka, and Takanori Isobe. Tweakable TWINE: building a tweakable block cipher on generalized feistel structure. In *IWSEC*, volume 11689 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2019.

[156] Sanjay E. Sarma, Stephen A. Weis, and Daniel W. Engels. RFID systems and security and privacy implications. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 454–469. Springer, 2002.

[157] Yu Sasaki and Kazumaro Aoki. Finding preimages in full MD5 faster than exhaustive search. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 134–152. Springer, Heidelberg, April 2009.

[158] Yu Sasaki and Yosuke Todo. New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 185–215. Springer, Heidelberg, April / May 2017.

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

[160] Danping Shi, Siwei Sun, Yu Sasaki, Chaoyun Li, and Lei Hu. Correlation of quadratic boolean functions: Cryptanalysis of all versions of full \mathsf MORUS. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 180–209. Springer, 2019.

[161] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In Bart Preneel and Tsuyoshi Takagi, editors, *CHES 2011*, volume 6917 of *LNCS*, pages 342–357. Springer, Heidelberg, September / October 2011.

[162] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *FSE 2007*, volume 4593 of *LNCS*, pages 181–195. Springer, Heidelberg, March 2007.

[163] Ling Sun, Wei Wang, and Meiqin Wang. More accurate differential properties of LED64 and midori64. *IACR Trans. Symmetric Cryptol.*, 2018(3):93–123, 2018.

[164] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the search of differential and linear characteristics with the SAT method. *IACR Trans. Symmetric Cryptol.*, 2021(1):269–315, 2021.

[165] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, DES(L) and other bit-oriented block ciphers. In *ASIACRYPT (1)*, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014.

[166] Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, Heidelberg, December 2014.

[167] Tomoyasu Suzaki and Kazuhiko Minematsu. Improving the generalized Feistel. In Seokhie Hong and Tetsu Iwata, editors, *FSE 2010*, volume 6147 of *LNCS*, pages 19–39. Springer, Heidelberg, February 2010.

[168] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. TWINE : A lightweight block cipher for multiple platforms. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 339–354. Springer, Heidelberg, August 2013.

[169] The ZUC design team. The ZUC-256 Stream Cipher. `http://www.is.cas.cn/ztzl2016/zouchongzhi/201801/W020180126529970733243.pdf`, 2018.

[170] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 287–314. Springer, Heidelberg, April 2015.

[171] Yosuke Todo and Masakatu Morii. Bit-based division property and application to simon family. In Thomas Peyrin, editor, *FSE 2016*, volume 9783 of *LNCS*, pages 357–377. Springer, Heidelberg, March 2016.

[172] James Waldrop, Daniel W. Engels, and Sanjay E. Sarma. Colorwave: an anticollision algorithm for the reader collision problem. In *ICC*, pages 1206–1210. IEEE, 2003.

[173] Lei Wang, Jian Guo, Guoyan Zhang, Jingyuan Zhao, and Dawu Gu. How to build fully secure tweakable blockciphers from classical blockciphers. In *ASIACRYPT (1)*, volume 10031 of *Lecture Notes in Computer Science*, pages 455–483, 2016.

[174] Louis Wingers. Supercop:supercop-20190110/crypto_stream/simon128128ctr/avx2. https://bench.cr.yp.to/supercop/supercop-20190110.tar.xz, 2019.

[175] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography*

- *SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2013.

[176] Wenling Wu and Lei Zhang. LBlock: A lightweight block cipher. In Javier Lopez and Gene Tsudik, editors, *ACNS 11*, volume 6715 of *LNCS*, pages 327–344. Springer, Heidelberg, June 2011.

[177] 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 Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 648–678. Springer, Heidelberg, December 2016.

[178] Jing Yang, Thomas Johansson, and Alexander Maximov. Spectral analysis of ZUC-256. *IACR Trans. Symmetric Cryptol.*, 2020(1):266–288, 2020.

[179] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. Impossibility and optimality results on constructing pseudorandom permutations (extended abstract). In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 412–422. Springer, Heidelberg, April 1990.

[180] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. On the construction of block ciphers provably secure and not relying on any unproved hypotheses. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 461–480. Springer, Heidelberg, August 1990.