Design and Cryptanalysis of Lightweight Symmetric Key Primitives

by

Raghvendra Rohit

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner	Carlisle Adams Professor, EECS, University of Ottawa
Supervisor	Guang Gong Professor, ECE, University of Waterloo
Internal Member	Anwar Hasan Professor, ECE, University of Waterloo
Internal Member	Mahesh Tripunitara Professor, ECE, University of Waterloo
Internal-external Member	Alfred Menezes Professor, C&O, University of Waterloo

Author's Declaration

Statement of Contributions

I am the sole author of the Chapters 1, 2, 9, 10 and 11. Some parts in Chapter 2 are adopted from *Understanding cryptography: A textbook for students and practitioners* [108] and *Handbook of applied cryptography* [82]. Chapters 3, 4, 5, 6, 7 and 8 contain materials from the papers or technical reports [14, 16, 15, 7, 8, 13, 12, 115, 116, 117] which I authored or co-authored. The exact contributions are explicitly mentioned at the beginning of each chapter.

Abstract

The need for lightweight cryptographic primitives to replace the traditional standardized primitives such as AES, SHA-2 and SHA-3, which are unrealistic in constrained environments, has been anticipated by the cryptographic community for over a decade and half. Such an anticipation came to reality by the apparent proliferation of Radio Frequency Identifiers (RFIDs), Internet of Things (IoT), smart devices and sensor networks in our daily lives. All these devices operate in constrained environments and require reasonable efficiency with low implementation costs and sufficient security. Accordingly, designing lightweight symmetric key cryptographic primitives and analyzing the state-of-the-art algorithms is an active area of research for both academia and industry, which is directly followed by the ongoing National Institute of Standards and Technology's lightweight cryptography (NIST LWC) standardization project. In this thesis, we focus on the design and security analysis of such primitives.

First, we present the design of four lightweight cryptographic permutations, namely sLiSCP, sLiSCP-light, ACE and WAGE. At a high level, these permutations adopt a Nonlinear Feedback Shift Register (NLFSR) based design paradigm. sLiSCP, sLiSCP-light and ACE use reduced-round Simeck block cipher, while WAGE employs Welch-Gong (WG) permutation and two 7-bit sboxes over the finite field $\mathbb{F}_{2^7}$ as their underlying nonlinear components. We discuss their design rationale and analyze the security with respect to differential and linear, integral and symmetry based distinguishers using automated tools such as Mixed Integer Linear Programming (MILP) and SAT/SMT solvers.

Second, we show the applications of these permutations to achieve Authenticated Encryption with Associated Data (AEAD), Message Authentication Code (MAC), Pseudorandom Bit Generator (PRBG) and Hash functionalities. We introduce the idea of the unified round function, which, when combined in a sponge mode can provide all the aforementioned functionalities with the same circuitry. We give concrete instantiations of several AEAD and hash schemes with varying security levels, e.g., 80, 96, 112 and 128 bits. Next, we present Spoc, a new AEAD mode of operation which offers higher security guarantees compared to traditional sponge-based AEAD schemes with smaller states. We instantiate Spoc with sLiSCP-light permutation and propose another two lightweight AEAD algorithms. Notably, 4 of our proposed schemes, namely ACE, Spix, Spoc and WAGE are round 2 candidates of NIST's LWC project.

Finally, we present cryptanalytic results on some lightweight ciphers. We first analyze the nonlinear initialization phase of WG-5 stream cipher using the division property based cube attack, and give a key recovery attack on 24 (out of 64) rounds with data and time complexities $2^{6.32}$ and $2^{76.81}$, respectively. Next, we propose a novel property of block ciphers called correlated sequences and show its applications to meet-in-the-middle attack. Consequently, we give the best key recovery attacks (up to 27 out of 32 rounds in a single key setting) on Simon and Simeck ciphers with block and key sizes 32 and 64 bits, respectively. The attack requires 3 known plaintext-ciphertext pairs and has a time complexity close to average exhaustive search. It is worth noting that variants of WG-5 and Simeck are the core components of aforementioned AEAD and hash schemes. Lastly, we present practical forgery attacks on Limdolen and HERN which are round 1 candidates of NIST LWC project. We show the existence of structural weaknesses which could be exploited to forge any message with success probability of 1. For Limdolen, we require the output of a single encryption query while for HERN we need at most 4 encryption queries for a valid forgery. Following our attack, both designs are eliminated from second round.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor Professor Guang Gong. This thesis would not have been possible without her endless support and encouragement throughout the last 3 years. Her knowledge and expertise in cryptography and communications introduced me to the big world of symmetric key cryptography, of which I had little knowledge before joining the PhD. She gave me the complete freedom to choose my own topics, helped me to overcome difficult situations, gave the opportunities to lead big projects like NIST LWC and grant applications for Compute Canada. Besides academics, she has always been a close friend to me. The knowledge and experience I have gained by being her student is something I will cherish for my entire life.

I am honoured and grateful to Professor Carlisle Adams, Professor Anwar Hasan, Professor Mahesh Tripunitara and Professor Alfred Menezes for serving as the committee members of my thesis. Their guidance and assistance helped in improving the thesis to a great extent.

I am thankful to Dr. Riham AlTawy and Dr. Kalikinkar Mandal for being wonderful colleagues and friends of mine. I am indebted to them for introducing me different attack techniques. The discussions which we had almost everyday extended my knowledge of cryptanalysis.

I would like to thank Professor Mark Aagaard and Nusa Zidaric for being great teammates and helping me to understand cryptographic hardware implementations. A big thanks to Nusa for offering a variety of tea almost everyday which motivated to have good conversations and work for long time at office.

I am grateful to all my collaborators including Professor Mark Aagaard, Dr. Riham AlTawy, Morgan He, Ashwin Jha, Dr. Kalikinkar Mandal, Professor Mridul Nandi, Dr. Gangqiang Yang, Yunjie Yi and Nusa Zidaric. The research contributions I had would not be possible without their help. I am also thankful to the former and current members of ComSec lab including Radi Abubaker, Ahmed Ayoub, Yao Chen, Chuyi Liu, Dongxu Ma, Khaled Nassar, Marat Sattarav, Amir Vakili, Bo Yang, Meng Yang and Jenny Yu, with whom I always had fruitful discussions. The work environment felt like home with their presence.

I have been fortunate enough to have very good friends both in and outside of the University of Waterloo who are close to me and made my life enjoyable. The list is never-ending, so

I won't mention any names. You guys were always supportive and helped me in all the ups and downs of my life.

Last but not the least, I am deeply grateful to my parents for their patience, inspiration and endless support over the past years I stayed away from home. My siblings always remained a great source of inspiration to me. I would not have been at this stage without their sacrifices.

A special thanks to NSERC, Compute Canada and NIST for supporting my research.

Dedication

To my mother.

Contents

List of Figures

List of Tables

List of Abbreviations

AEAD	Authenticated Encryption with Associated Data
ANF	Algebraic Normal Form
CAESAR	Competition for Authenticated Encryption: Security, Applicability, and Robustness
GFS	Generalized Feistel Structure
LFSR	Linear Feedback Shift Register
LCSC	Linear Characteristic Square Correlation
LHSC	Linear Hull Square Correlation
MAC	Message Authentication Code
MEDCP	Maximum Expected Differential Characteristic Probability
MEDP	Maximum Estimated Differential Probability
MELCSC	Maximum Expected Linear Characteristic Square Correlation
MELHSC	Maximum Estimated Linear Hull Square Correlation
MILP	Mixed Integer Linear Programming
NLFSR	Nonlinear Feedback Shift Register
PRBG	Pseudorandom Bit Generator
SAT	Satisfiability
SC	Square Correlation
SMT	Satisfiability Modulo Theories
WG	Welch Gong
WGP	Welch Gong Permutation

Chapter 1

Introduction

Contents

1.1 Motivation

Over the past few years, there has been an overwhelming surge in development of Internet of Things (IoT) including Radio Frequency Identifiers (RFIDs), smart devices and sensor networks. The IoT connects an extraordinarily wide spectrum of devices ranging from personal computers to remote servers to smart devices (e.g., smart watches, smart speakers, home automation) to embedded systems. The RFIDs such as Electronic Product Code (EPC) tags [6] are typically used in identification of market products, while sensor networks monitor the physical environment (e.g., weather forecast, industrial process on site) by connecting a network of sensors to a central hub. It is expected that the number of such devices will be more than 20 billion by 2020 with an approximate market value of 7.1 trillion US dollars [75].

All these devices collect and transmit a huge volume of data which may risk the privacy of users. Therefore, attaining proper security goals is the primary requirement. In particular, the transmitted data needs to be encrypted and/or authenticated. Additionally, such devices operate in varying environments and thus have different resource and performance requirements. For example, EPC tags [6, 80] are highly constrained in terms of physical implementation area and power consumption, while vehicular embedded systems require a very low latency and real-time response [84]. Sensor networks mostly operate on low power batteries (e.g., solar energy). For IoTs, low latency and/or high throughput are required. Accordingly, the three metrics, namely 1) *security and functionalities*: encryption, authentication or both; 2) *resource constraints*: area and code size (in case of software); 3) *performance*: latency, power consumption, throughput and RAM size, define *lightweight crytpography* which aims to find a balance of trade-off among them. In simple words, lightweight cryptography is about designing secure cryptographic primitives for resource constrained applications.

Designing an algorithm with optimal metrics for constrained environments is a challenging task. The current National Institute of Standards and Technology (NIST) approved standards such as AES [102], SHA-2 [103] and SHA-3 [104] perform well in desktop and server environments but not in resource constrained devices. For example, AES round-based hardware cost is too high in Gate Equivalents (GE). The hash functions SHA-2 and SHA-3 with state sizes 512 and 1600 bits have large memory and area requirements. In this context, designing algorithms which fit the resource constrained scenarios and can outperform the current standards remains an active area of research. Starting from the eSTREAM project [4] to Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [1] to the ongoing NIST lightweight cryptography (NIST LWC) standardization project [5], numerous lightweight symmetric key primitives such as stream ciphers, block ciphers, hash functions and authenticated encryption with associated data (AEAD) algorithms have been proposed.

The stream ciphers expand a fixed length secret key to long keystream sequence using a Pseudorandom Bit Generator (PRBG) based on Nonlinear Feedback Shift Register (NLFSR) design paradigm. Examples of lightweight stream ciphers include Grain [73], Trivium [53], Micky [20], WG [101], Plantlet [99] and Lizard [72]. Block ciphers take an n-bit plaintext and κ-bit ($\kappa \geq n$) secret key as input and output a random looking n-bit ciphertext after iterating the round function multiple times. Such a round function can be designed using the two well known constructions, namely Feistel structure and Substitution Permutation Network (SPN). Examples of lightweight block ciphers with a Feistel round are TEA [132], Hight [74], Twine [125], Simon and Speck [23] and Simeck [135], while LED [71], PRESENT [42], EPCBC [136], Prince [43], Skinny [25], GIFT [21] and Craft [26] are based on an SPN round function.

A cryptographic hash function takes an arbitrary length message as an input and output a short fixed length *fingerprint* of the message. Their constructions rely on stream ciphers, block ciphers or cryptographic permutations. Examples of lightweight hash functions that utilize cryptographic permutation encompass Keccak (smaller versions) [37], Photon [70], Quark [17], Spongent [41] and Gimli [29]. On the other hand, an AEAD scheme provides integrity and authenticity in addition to confidentiality. Such schemes are constructed by using stream ciphers, block ciphers or cryptographic permutations in a mode, e.g., GCM [96], OCB [114], OCB3 [88] and sponge [31]. A majority of the NIST LWC round 2 candidates are based on sponge mode or its variants. A few examples are Ascon [60], ACE [7], Gimli [29], PHOTON-Bettle [22], SPARKLE [24], SPIX [13], SpoC [12], Spook [28], Subterranean 2.0 [51], WAGE [8] and Xoodyak [50].

Although lightweight, the aforementioned schemes vary in hardware and software performances because of structural differences. In particular, the choice of the round function, i.e., Feistel or SPN, together with the number of rounds determine the performance. A Feistel round is cheaper in hardware than an SPN round but has slower diffusion. Low number of rounds is good for high throughput but may not guarantee security against generic attacks. Thus, there exist a multitude of options which needs to be analyzed thoroughly for a lightweight design.

1.2 Thesis Contributions

This thesis presents research contributions in the area of lightweight cryptography including both the design and cryptanalysis. We push the design limits to low hardware area and power consumption with efficient software performance. Consequently, we propose the design of four lightweight cryptographic permutations: sLiSCP, sLiSCP-light, ACE and WAGE. Another aspect is the design of '*one-for-all*' unified round function in a sponge mode which can provide multiple cryptographic functionalities with a cheap hardware overhead (extra cost for control logic only). Our proposed AEAD and hash algorithms have better or comparable performance relative to existing symmetric key primitives at the same security level. Notably, four of the NIST LWC round 2 candidates, namely ACE, Spix, Spoc and WAGE are the contribution of this thesis. From a cryptanalysis perspective, a novel contribution is the idea of correlated sequences and their applications to meet-in-the-middle attack, which resulted in best key recovery attacks on NSA's cipher Simon-32/64 and its variant Simeck-32/64. In addition, we find forgery attacks on NIST LWC round 1 candidates Limdolen and HERN. As a result, these algorithms are removed from the second round of the competition.

1.3 Outline

The rest of the thesis is organized as follows. Chapter 2 provides the preliminaries and related work. We give the mathematical description of symmetric key primitives and generic cryptanalysis techniques. We also discuss the metrics that define lightweight cryptography and introduce our design principles.

The thesis is then divided into three parts: 1) Design of lightweight cryptographic permutations (Chapters 3-5), 2) Mode of operations for AEAD and Hash (Chapters 6 and 7), and 3) Cryptanalysis of lightweight symmetric key primitives (Chapters 8-10).

In Chapter 3, we propose sLiSCP and sLiSCP-light family of permutations which utilize unkeyed reduced-round Simeck block cipher as their building block. We provide an in-depth security analysis of Simeck, and hence sLiSCP and sLiSCP-light, using automated tools. Chapter 4 introduces ACE which is a generalized version of sLiSCP and sLiSCP-light. We present its detailed security analysis along with the design rationale. In Chapter 5, we propose WAGE, a lightweight permutation based on the WG family of stream ciphers. We show that simple tweaks can transform WAGE to the original WG cipher. We further discuss its security analysis and justify our design choices.

Chapter 6 introduces the idea of the unified round function for a sponge mode. We show its adaptability to multiple cryptographic functionalities with a low hardware overhead. We give the generic descriptions of AEAD and hash algorithms, and present several instances of these algorithms with varying security levels using sLiSCP, sLiSCP-light, ACE and WAGE permutations. In Chapter 7, we present a new AEAD mode of operation called Spoc, and its two instantiation using sLiSCP-light.

Chapter 8 presents a key recovery attack on the reduced-round WG-5 stream cipher using the division property based cube attacks. In Chapter 9, we propose a novel property of block ciphers called correlated sequences. We show their applications to meet-in-the-middle attacks, and then provide key recovery attacks (27 out of 32 rounds) on two lightweight block ciphers Simon-32/64 and Simeck-32/64 in a single key setting. Chapter 10 presents the practical forgery attacks on two NIST LWC round 1 candidates Limdolen and HERN.

Finally, Chapter 11 concludes the thesis with possible future research directions.

Chapter 2

Preliminaries

Contents

2.1 Notation

We denote by $\mathbb{F}_2$ the finite field consisting of $\{0, 1\}$. For a positive integer n, $\mathbb{F}_2^n$ and $\mathbb{F}_{2^n}$ denote the n dimensional vector space over $\mathbb{F}_2$ and an extension field over $\mathbb{F}_2$ defined using a irreducible polynomial over $\mathbb{F}_2$ of degree n, respectively. Let C_s represent the coset modulo $2^n - 1$, i.e., $C_s = \{s, 2s, \ldots, 2^{n_s-1}s\}$ where n_s is the smallest number such that $s \equiv 2^{n_s}s \bmod 2^n - 1$, and s is the smallest number in C_s, and denoted as the coset leader.

We use $\{0, 1\}^n$, $\{0, 1\}^\star$ and ϵ to denote the set of all length n, variable length and empty bitstrings, respectively. For any string $X \in \{0, 1\}^\star$, $|X|$ denotes the length of X in bits and by $(X_0, \cdots, X_{l-1}) \overset{n}{\leftarrow} X$ we refer to the n-bit block parsing of X where $|X_i|= n$ for $0 \leq i \leq l - 2$ and $1 \leq |X_{l-1}|\leq n$. By $X \overset{\$}{\leftarrow} \{0, 1\}^n$, we mean a random n bitstring drawn from $\{0, 1\}^n$. We write X in bits as $(x_0, x_1, \cdots, x_{|X|-1})$. For strings X and Y, the operations $X \odot Y, X \oplus Y, X|Y, X||Y$ and $< X, Y >$ denote the bitwise AND, XOR, OR, concatenation and scalar product of X and Y, respectively. Moreover, $\mathsf{L}(\cdot)$ denotes the left cyclic shift operator, i.e., for $x \in \{0, 1\}^n$, $\mathsf{L}^i(x) = (x_i, x_{i+1}, \ldots, x_{n-1}, x_0, x_1, \ldots, x_{i-1})$.

We use $\mathsf{Pr}(X = x)$ to denote the probability that a random variable X equals x. By $\mathsf{Pr}(X_0 = x_0, X_1 = x_1, \ldots, X_{l-1} = x_{l-1})$ we refer to the joint probability of random variables $X_0, X_1, \ldots, X_{l-1}$ taking values $x_0, x_1, \ldots, x_{l-1}$, respectively. $\mathsf{Pr}(A|B)$ denotes the conditional probability of A given B. Furthermore, the symbol $X \sim U(\{0,1\}^n)$ denotes that X follows a uniform distribution over $\{0,1\}^n$. Finally, the secret key space is denoted by $\mathcal{K}$.

Note that we use '+' and '$\oplus$' and '$\bigoplus$' interchangeably throughout this chapter, if the meaning is clear in context.

2.2 Modern Cryptography

Cryptography, the word derived from two Greek words *kryptos* meaning "hidden secret" and *graphein* meaning "to write", is the science of secret communications in the presence of malicious adversaries. The history dates back to 2000 B.C with its initial use as secret hieroglyphics by Egyptians [81]. Other historic examples include Caesar and Vignere ciphers, and German Enigma used in World War II. Although the former ciphers could preserve the secrecy, they got attacked with the advent in computing power. Cryptography has now evolved to *modern cryptography* which is based on hard mathematical problems such as integer factorization, discrete logarithm or one-way functions, i.e., recovering a secret key which lies in an exponential search space.

In the following, we give a high level overview of the fundamental problem in modern cryptography and its major branches.

2.2.1 The fundamental cryptographic problem and goals

Consider two parties Alice and Bob[1]. Let's say Alice has a message x which she wants to send to Bob over the insecure channel, e.g., internet. If Alice sends x as it is to Bob (Figure 2.1), then an adversary Eve knows the message in plaintext. Eve could further change x to x' and then send x' to Bob. In this case, Bob is unaware of the modification and cannot ensure whether the message is coming from Alice or Eve. Furthermore, on receiving the exchanged message, Alice or Bob cannot deny it. Thus, the fundamental problem in modern cryptography is to address these issues and achieve secure communication over a insecure channel. More precisely, a secure communication requires the following four goals.

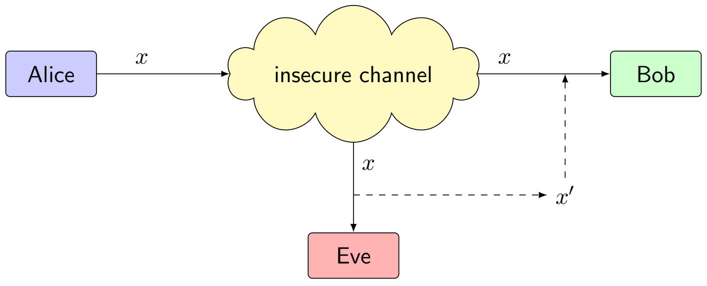


Figure 2.1: Communication over insecure channel

[1]In real world Alice and Bob could be laptops, smartphones, servers or IoT devices.

1. Confidentiality. For a passive adversary, i.e., an adversary who is only listening over the channel, it must be infeasible to learn any information about the message x. In particular, no one except the sender (Alice) and receiver (Bob) learns x.

2. Integrity. It must be infeasible for an active adversary, i.e., an adversary who modifies the message x to x', to have x' be accepted. In other words, any changes in the exchanged messages are easily detectable by Alice or Bob.

3. Authenticity. For an active adversary, it must be infeasible to impersonate the identity of sender or receiver by modifying the message or creating a complete new message. In particular, Alice and Bob should be able to detect that the exchanged messages are originating from either of them or from an adversary. Note that there is a subtle difference between integrity and authenticity. For integrity, any changes in the message should be detectable, while authenticity requires that message is exactly the same as it was sent.

4. Non-repudiation. For the above three properties, we have considered that Alice and Bob are honest and Eve is the adversary. What if either of Alice or Bob acts as an adversary? In this case, we require that a denial in the commitment of exchanged messages by either of them is easily detectable. Note that this property can only be achieved by asymmetric cryptography.

2.2.2 Branches of cryptography

Cryptography is mainly divided into *symmetric cryptography* and *asymmetric cryptography* (Figure 2.2), based on which party/parties hold the secret key. A third branch often termed as *crytographic protocols* use the former two cryptographic approaches as building blocks. We now describe each of the them in detail as follows.

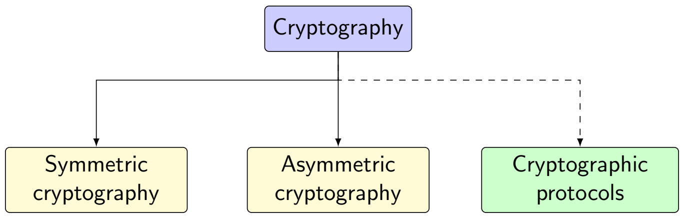


Figure 2.2: Branches of cryptography

2.2.2.1 Symmetric cryptography

As the name suggests, both sender and receiver hold the same secret key k. To send a message x, Alice uses an encryption function Enc which takes x and k as input and output a random looking y. Alice then sends y to Bob over the insecure channel. On receiving y, Bob computes the decryption function $\mathsf{Dec}(y, k)$ which returns the message x as the output. The entire procedure is depicted in Figure 2.3. In addition to classical encryption, symmetric key cryptography can also provide data integrity and authentication (as described later in Section 2.3).

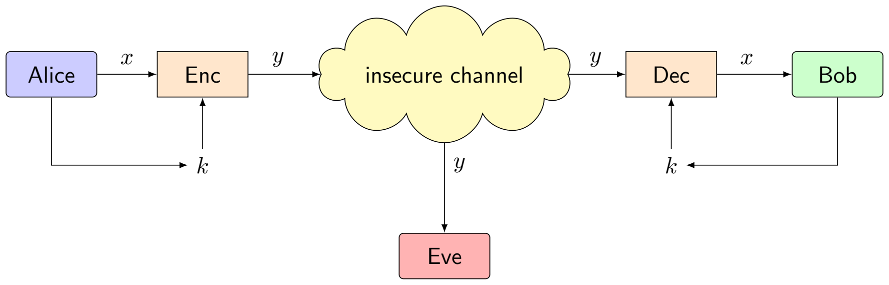


Figure 2.3: Symmetric key encryption and decryption

Remark 2.1. The shared secret key could be obtained via some secure channel. In real applications, it is done through a combination of asymmetric cryptography and certificates (e.g., Diffie Hellman key exchange protocol [56] where public keys are certified by a trusted CA). From now onwards, when we talk about symmetric cryptography, we assume that only Alice and Bob know the secret key independent of how they obtain it.

2.2.2.2 Asymmetric cryptography

In asymmetric or public-key cryptography, a user posses two keys, i.e., (pk, sk), namely *public key pk* and *private key sk*. For a typical encryption such as RSA, Alice uses Bob's public key pk_{Bob} to encrypt a message. Bob then uses his private key sk_{Bob} for decryption. This is illustrated in Figure 2.4. In addition to classic encryption, asymmetric key cryptography can be used for key establishment and digital signatures[2].

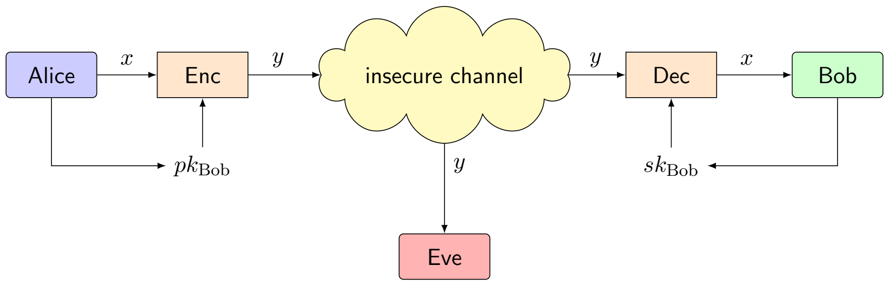


Figure 2.4: Asymmetric (public-key) encryption and decryption

[2]Non-repudiation can be achieved through digital signatures as private key is only known to the user who signs the message.

Remark 2.2. The hardness in asymmetric key cryptography is based on hard computational problems such as integer factorization or discrete logarithm. For symmetric cryptography, it is based on recovering the secret key which has an exponential search space in the length of key.

2.2.2.3 Cryptographic protocols

Roughly speaking, a cryptographic protocol is a set of rules which uses symmetric and asymmetric primitives as building blocks for secure communication. The Transport Layer Security (TLS) protocol used in every modern-day web browser is such an example. It initially uses asymmetric approach for key establishment, and then utilizes symmetric cryptography with established keys for exchanging encrypted (and/or authenticated) messages (Chapter 8 [48]).

2.3 Symmetric Key Primitives

Symmetric key primitives are widely deployed cryptographic primitives in day-to-day life. They have a wide range of applications, e.g., secure communications such as TLS 1.2/1.3, IPSec and SSL, online and chip based payments, WIFI and sensor networks or RFIDs. Such primitives typically offer three functionalities: confidentiality, data integrity and authentication. The major symmetric key primitives which offer these functionalities include *stream ciphers*, *block ciphers*, *hash functions*, *message authentication codes* and *authenticated encryption with associated data algorithms*. In the following, we give a generic description of these primitives.

2.3.1 Stream ciphers

A stream cipher encrypts/decrypts a message bit by bit (or byte by byte). The idea is to expand a secret key with fixed length to large keystream. Mathematically, it is a function $\mathcal{F}$ which takes a κ-bit secret key K and an n-bit initialization vector IV as input and output a sequence of keystream bits $z_0, z_1, \ldots$ (Figure 2.5). The encryption and decryption is simply done using bitwise XOR operation and given by

$$\text{Encryption: } c_i = m_i \oplus z_i$$
$$\text{Decryption: } m_i = c_i \oplus z_i$$

Assumption. For a fixed K, the public value IV should never be repeated otherwise the keystream repeats and the confidentiality is completely lost.

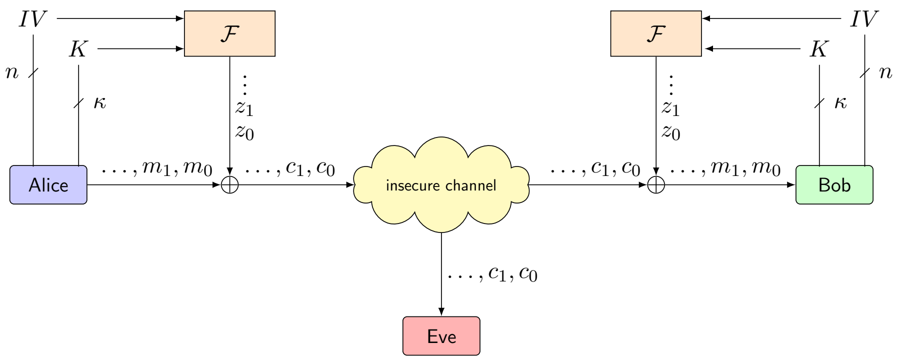


Figure 2.5: Schematic of stream cipher encryption and decryption

Properties. The security properties of keystream highly depends on the choice of $\mathcal{F}$. In general, the following properties are needed.

1. *Indistinguishability:* It should be infeasible to distinguish an l-bit keystream from a randomly drawn l-bit string from $\{0,1\}^l$.

2. *Unpredictibility:* Given l bits of keystream $z_0, \dots, z_{l-1}$, the difference between $\mathsf{Pr}(z_l = 0)$ and $\mathsf{Pr}(z_l = 1)$ is negligible.

Sometimes we also require that keystream has a *long period*. Since it is difficult to guarantee a lower bound on the period (especially when $\mathcal{F}$ is an NLFSR [67, 68]), this property is desirable but not easy to achieve.

Working procedure of $\mathcal{F}$. $\mathcal{F}$ typically operates in two phases: 1) *Key Initialization Phase (KIA)* and 2) *Keystream Generation Phase (KSG)*. Both phases are explained below.

1. KIA phase: The state is first loaded with K, IV and some constants. Next, we update the state for R rounds without producing any output.

2. KSG phase: Output the keystream bit and update the state. Repeat till the required number of keystream bits are obtained.

Examples of $\mathcal{F}$. A trivial example of $\mathcal{F}$ is an m-stage Linear Feedback Shift Register (LFSR) defined using a degree m primitive poynomial over $\mathbb{F}_2$. It guarantees a period of $2^m - 1$, however such constructions are easily attacked using Berlekamp-Massey algorithm with the knowledge of only $2m$ keystream bits. Other well known examples of $\mathcal{F}$ include Grain [73], Trivium [53], WG [101], Acorn [133], Lizard [72] and Plantlet [99] which are based on NLFSR or a combination of LFSR and NLFSR with a filtering function.

2.3.2 Block ciphers

A block cipher is a deterministic algorithm which encrypts/decrypts a block of bits at a time. In particular, it is a combination of two algorithms, an encryption algorithm E and the decryption algorithm E^{-1}. Let K be a κ-bit secret key, and P and C denote the n-bit plaintext and ciphertext, respectively. Furthermore, assume $\kappa \geq n$. A pictorial illustration of block cipher is shown in Figure 2.6.

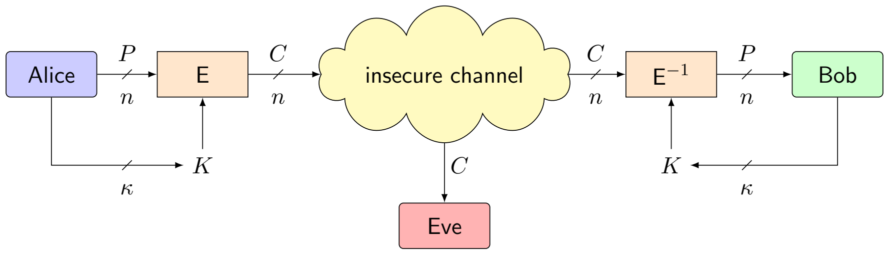


Figure 2.6: Schematic of block cipher encryption and decryption

An encryption algorithm E is a function $\mathsf{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ such that for each $K \in \{0,1\}^\kappa$, the function $\mathsf{E}_K : \{0,1\}^n \to \{0,1\}^n$ is a permutation. The decryption algorithm is symmetrical to the encryption algorithm. In addition, for correctness it should satisy $\mathsf{E}_K^{-1}(\mathsf{E}_K(P)) = P$ for all $P \in \{0,1\}^n$ and $K \in \{0,1\}^\kappa$.

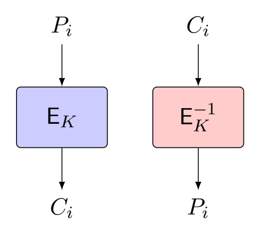
Remark 2.3. There exists $2^n!$ permutations which map n bits to n bits. A block cipher is small subset of this set with 2^κ permutations. In Chapters 3-5, we will look at the design of cryptographic permutations.

In Figure 2.6, we have shown the encryption and decryption procedures for a single block. However, in real applications, the message length is greater than n. There exist many ways of encrypting a long message by using a block cipher in a mode.

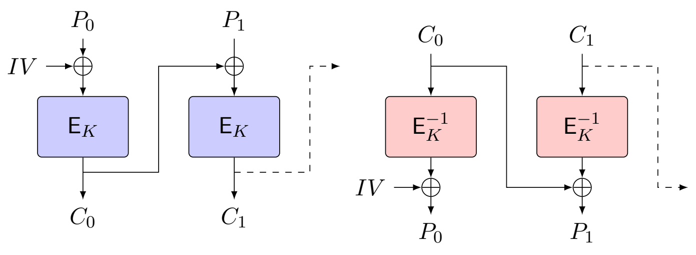
2.3.2.1 Block cipher modes

Let K be fixed, IV be an n-bit public initialization vector, and assume that the message length is always a multiple of block size n[3]. Figure 2.7 depicts the most commonly used block cipher modes where blue and red colored boxes denote the encryption and decryption, respectively. Note that OFB, CFB and counter modes are of stream cipher encryption using block ciphers and may use only a portion of the output of E_K depending on the length of the last plaintext block.

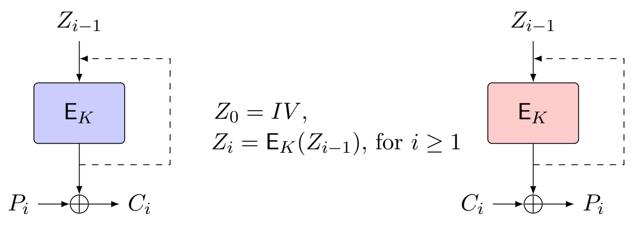
[3]If not, we do the padding



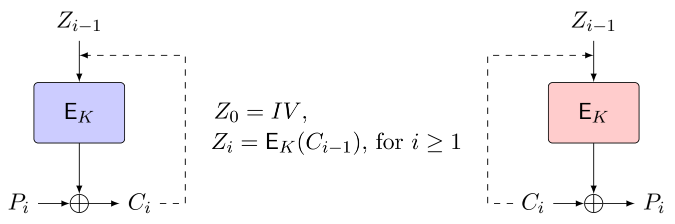
a) Electronic codebook mode (ECB)



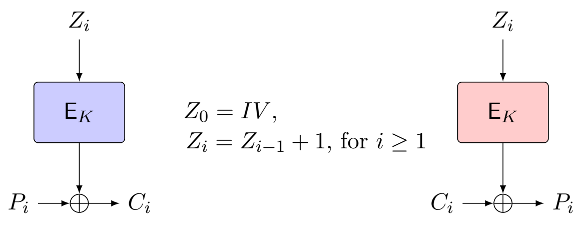
b) Cipher block chaining mode (CBC)



c) Output feedback mode (OFB)



d) Cipher feedback mode (CFB)



e) Counter mode

Figure 2.7: Block cipher modes of operation

2.3.2.2 Generic structures of block ciphers

Until now, we have seen the encryption and decryption procedures using a block cipher. We now look into the structural details of E_K and E_K^{-1}. As depicted in Figure 2.8, the encryption (resp. decryption) algorithm is an iterative permutation where the output is obtained after iterating the round function $\mathcal{RF}$ (resp. $\mathcal{RF}^{-1}$) R times. The round function takes n-bit intermediate stage and κ'-bit round key ($\kappa' \leq n$) as input and produces an n-bit output. The round keys k_i are derived from the master key K using a Key Scheduling Algorithm.

A typical design of round function is based on either Feistel or Substitution Permutation Network (SPN) which are described below.

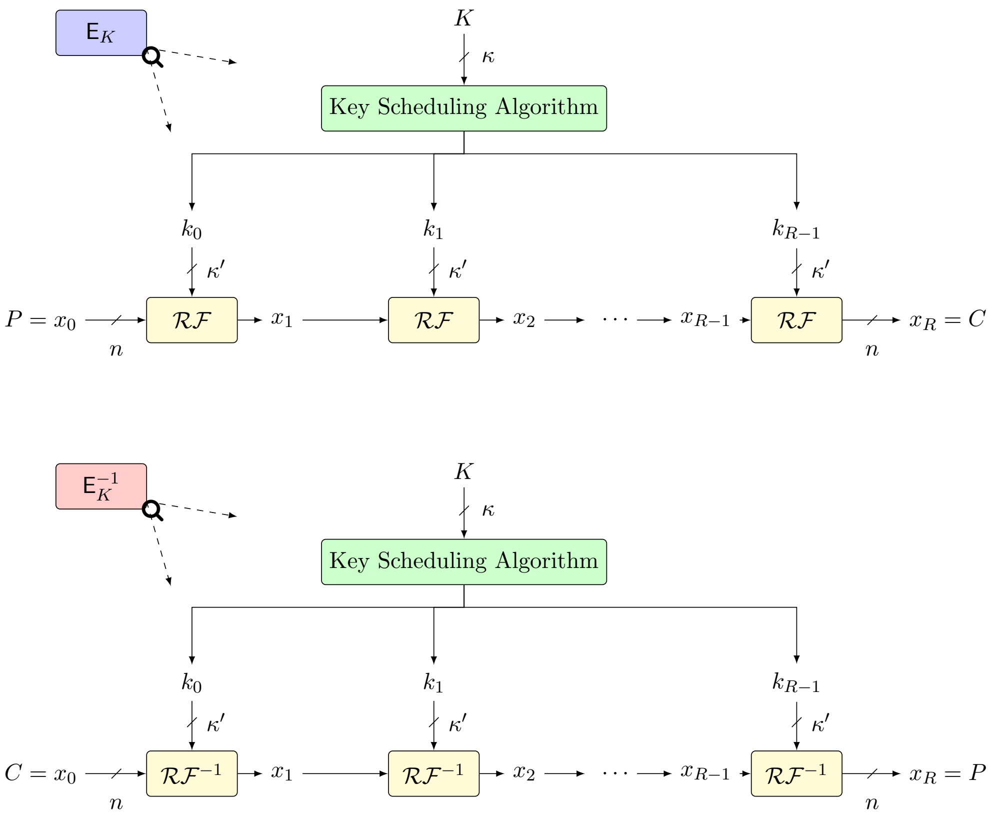


Figure 2.8: Block cipher as an iterative structure

Feistel round function. Roughly speaking, a Feistel round function process only half of the bits in a single round. Let $x_i = x_i^1 \| x_i^0$, x_i^1 and $x_i^0 \in \{0,1\}^{n/2}$, and $\mathsf{S}{:}\{0,1\}^{n/2} \to \{0,1\}^{n/2}$ be a nonlinear function. Then, a single round of Feistel network is shown in Figure 2.9 and computed as follows.

$$\begin{aligned} x_{i+1}^1 &\leftarrow \mathsf{S}(x_i^1) \oplus x_i^0 \oplus k_i, \\ x_{i+1}^0 &\leftarrow x_i^1. \end{aligned}$$

For a Feistel round function, S does not have to be invertible. Thus, encryption and decryption can be performed using the same round function by changing the order of round keys. Notable examples of such construction are Lucifer [65], DES [121], Simon [23] and Simeck [135]. This structure was later generalized into *Type I*, *Type II* and *Type III Generalized Feistel Structures* (GFS) [140, 106]. The classic DES is an example of Type I GFS, while the round function of Type II GFS with 4 branches is shown in Figure 2.10. Examples of GFS

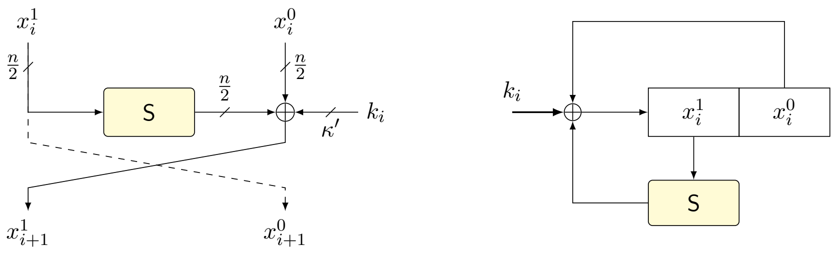


Figure 2.9: Feistel round function (as an NLFSR on right)

based ciphers include RC6 [112], MARS [44], CAST-256 [10], HIGHT [74], TWINE [125] and CLEFIA [119].

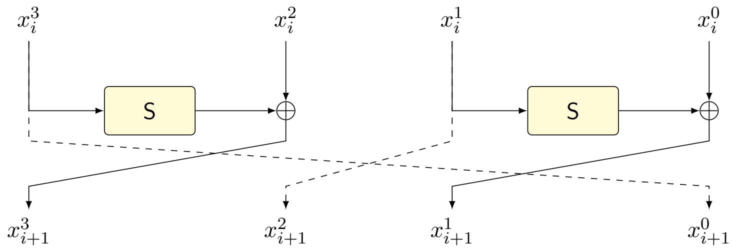


Figure 2.10: Type II GFS round function without round keys

SPN round function. Contrary to the Fesitel round, an SPN round function process all bits in a single round. It is basically a composition of 3 layers, namely 1) AddRoundKey, 2) Substitution layer and 3) Linear layer. The substitution layer divides the n-bit state into $\frac{n}{m}$ m-bit words and then applies the nonlinear invertible function S to each word[4]. The linear layer then mixes the bits of the state in a word or bit wise fashion (Figure 2.11).

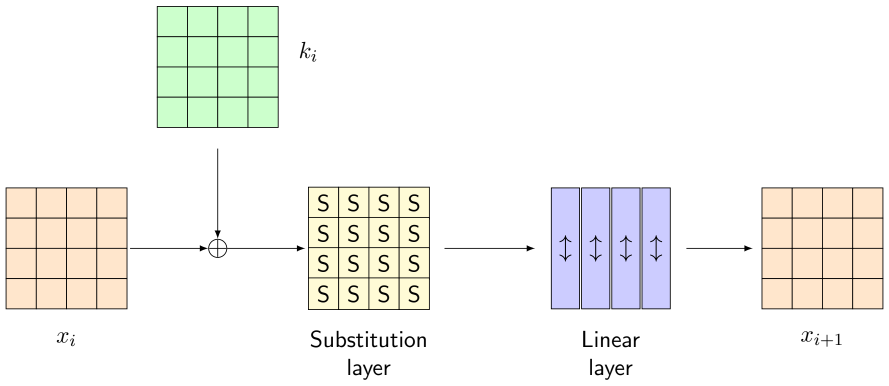


Figure 2.11: SPN round function

[4]Here S is a m-bit to m-bit mapping

Most notable example of an SPN round is the AES round function [52] where $n = 128$, $m = 8$ and linear layer is the combination of ShiftRows and MixColumns operations. Other examples with linear layer as bit permutation include lightweight block ciphers PRESENT [42] and GIFT [21] where $n = 64$ and $m = 4$.

2.3.2.3 Tweakable block ciphers

Roughly speaking, a block cipher with a public value (referred to as *tweak*) as an input in addition to plaintext and key is a *tweakable block cipher* (TBC) [91] (Figure 2.12). Mathematically, it is a function $\mathsf{E} : \{0,1\}^\kappa \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$ such that for all key and tweak pairs $(K, TK) \in \{0,1\}^\kappa \times \{0,1\}^t$, $\mathsf{E}_{(K,TK)} : \{0,1\}^n \to \{0,1\}^n$ is a permutation. The decryption algorithm is defined analogously.

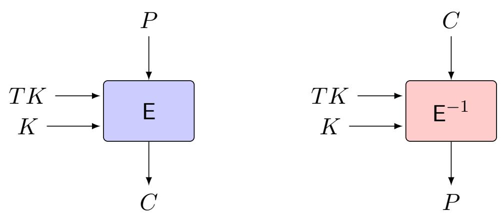


Figure 2.12: Tweakable block cipher

The modes of operation and underlying structures of round function is similar to classic block ciphers. The only difference is in the key scheduling algorithm which now gives tweakey tk_i as the output in each round. Note that only TK value is public and not tk_i's. Examples include Skinny [25], Deoxys-BC [78] and CRAFT [26].

2.3.3 Hash functions

A cryptographic hash is an unkeyed primitive which takes an arbitrary length message as an input and output a short fixed length *fingerprint* of the message. The fingerprint is often called as a *hash value* or *message digest*.

Security properties. A hash function $\mathcal{H} : \{0,1\}^\star \to \{0,1\}^n$ should satisy the following properties.

1. *Preimage resistance:* Given a message digest $y \in \{0,1\}^n$, it is infeasible to find x such that $\mathcal{H}(x) = y$.

2. *Collision resistance:* It is infeasible to find any two messages $x_1 \neq x_2$ such that $\mathcal{H}(x_1) = \mathcal{H}(x_2)$.

3. *Second preimage resistance:* Given x_1, and thus $\mathcal{H}(x_1)$, it is infeasible to find any x_2 such that $x_1 \neq x_2$ and $\mathcal{H}(x_1) = \mathcal{H}(x_2)$.

2.3.3.1 Merkle-Damgård construction

The Merkle-Damgård construction is a well known method to construct a hash function using a one-way compression function. Let f be a compression function and $M_0, \cdots, M_{l-1}$ be l blocks of message M obtained after padding. The hash value is computed as follows.

$$\begin{aligned} H_{-1} &= \mathsf{constant} \\ H_i &\leftarrow f(H_{i-1}, M_i), \quad \text{for } 0 \le i \le l-1 \\ \mathcal{H}(M) &= H_{l-1} \end{aligned}$$

This procedure is shown in Figure 2.13. Examples of such hash functions are MD5 [111] and SHA-1 [61].

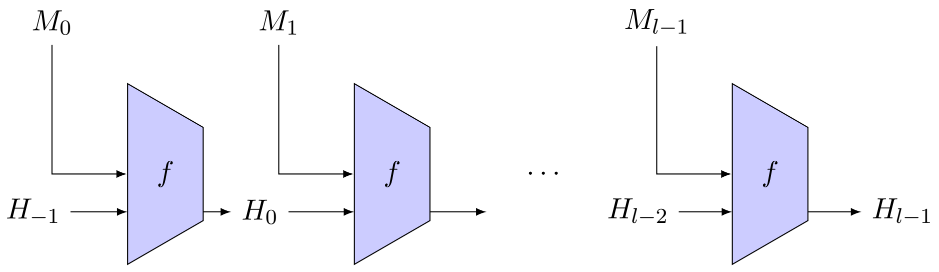


Figure 2.13: Merkle-Damgård hash construction

2.3.3.2 Hash functions from block ciphers

Block ciphers can be used in multiple ways to construct hash functions. Two such widely known constructions are Davies-Meyer and Miyaguchi-Preneel hash modes. In Davies-Meyer construction a message block is used as the key, while for Miyaguchi-Preneel mode the chaining value H_i is taken as the key. For both constructions the initial chaining value H_{-1} is a fixed constant and the last chaining value is taken as the message digest (Figure 2.14).

2.3.3.3 Sponge based hash

Sponge functions introduced by Bertoni *et al.* [31] take arbitrary length input and produce an arbitrary length output. Their underlying primitive is a b-bit unkeyed cryptographic permutation P where $b = r + c$ and r denotes the rate part of state, while c is the capacity. The message M after padding is divided into chunks of r-bit blocks which are then absorbed into the state r bits at a time. This phase is called as *absorbing phase*. After all the message blocks are processed, *squeezing phase* begins where r-bit digest is taken at a time until $l = \frac{c}{r}$ blocks are squeezed. Finally, the hash of message is given by $\mathcal{H}(M) = H_0 \| H_1 \| \cdots \| H_{l-1}$. The entire process is depicted in Figure 2.15.

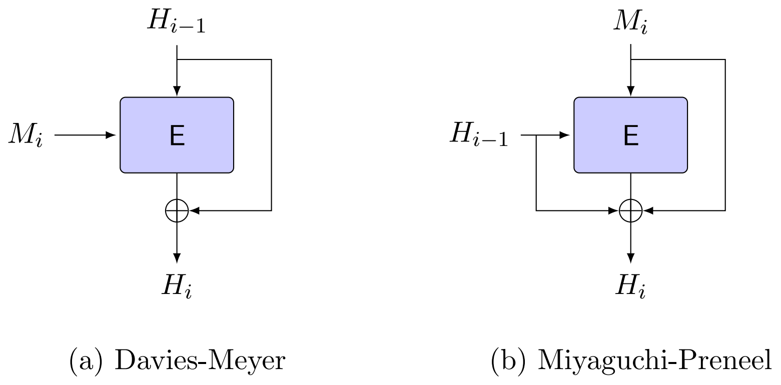


Figure 2.14: Block cipher based hash functions

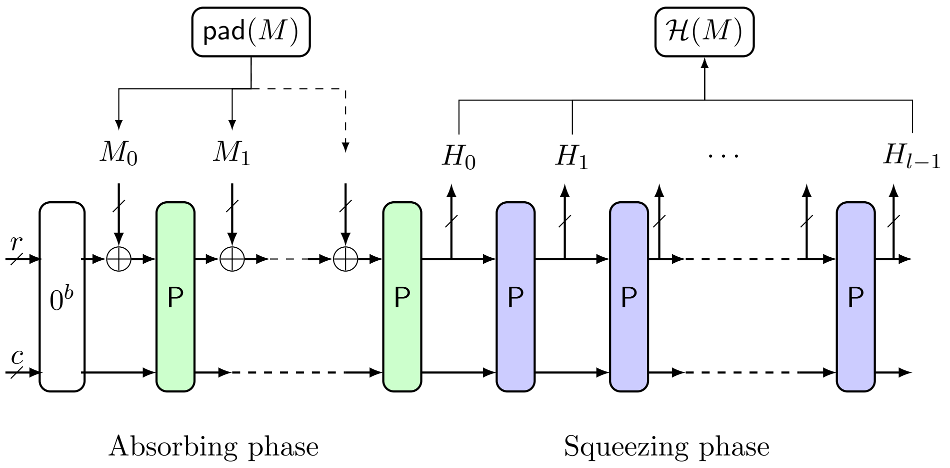


Figure 2.15: Sponge based hash function

Hash functions such as SHA-3 winner Keccak [32], Photon [70], Quark [17] and Spongent [41] are based on sponge construction with different permutations. In Chapter 6, we present new lightweight hash functions which have better or comparable performance with all the aforementioned hash functions at the same security level.

2.3.4 Message authentication codes

At a high level, a message authentication code (MAC) is a keyed hash function. A MAC function $\mathsf{MAC} : \{0,1\}^\star \times \{0,1\}^\kappa \rightarrow \{0,1\}^t$ takes arbitrary length message M and κ-bit secret key as input and output a t-bit tag T which authenticates the message M. The receiver with the knowledge of secret key and M computes the tag T'. If T' equals T, then only the message authentication is successful.

Examples of block cipher based MACs include CBC-MAC, PMAC [39] and CMAC [77], while HMAC [27] is a hash based MAC. To use sponge based hash (Figure 2.15) as a MAC, we load the state with the secret key and public initialization vector and then call the permutation P. Next, we absorb the message blocks and then extract the tag similarly to hash digest.

2.3.5 Authenticated encryption with associated data algorithms

The authenticated encryption with associated data algorithm (AEAD) provides confidentiality, integrity and authenticity at the same time. The AEAD algorithm AE is a combination of two algorithms, an authenticated encryption algorithm $\mathsf{AE}_{\mathsf{enc}}$ and the verified decryption algorithm $\mathsf{AE}_{\mathsf{dec}}$. A high level overview of AE is illustrated in Figure 2.16 and described below.

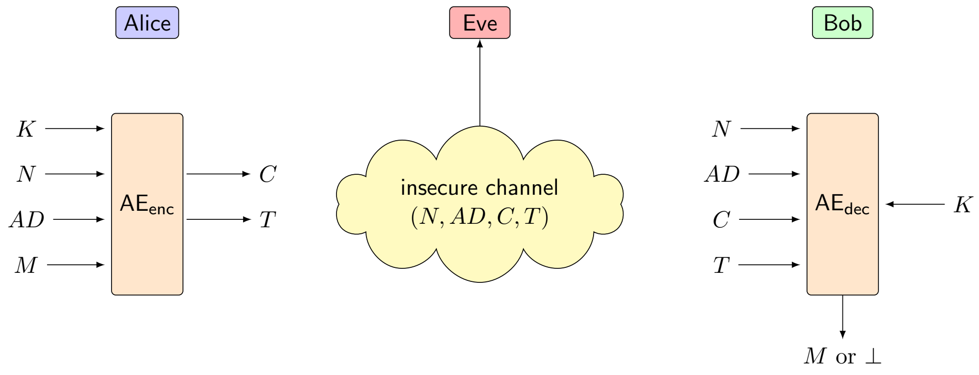


Figure 2.16: Schematic of AEAD algorithm

An authenticated encryption algorithm $\mathsf{AE}_{\mathsf{enc}}$ takes as input a secret key K of length κ bits, a public message number N (nonce) of size n bits, a block header AD (a.k.a, associated data) and a message M. The output of $\mathsf{AE}_{\mathsf{enc}}$ is an authenticated ciphertext C of the same length as M, and an authentication tag T of size t bits. Mathematically, $\mathsf{AE}_{\mathsf{enc}}$ is defined as

$$\mathsf{AE}_{\mathsf{enc}} : \{0,1\}^{\kappa} \times \{0,1\}^{n} \times \{0,1\}^{*} \times \{0,1\}^{*} \to \{0,1\}^{*} \times \{0,1\}^{t}$$

with

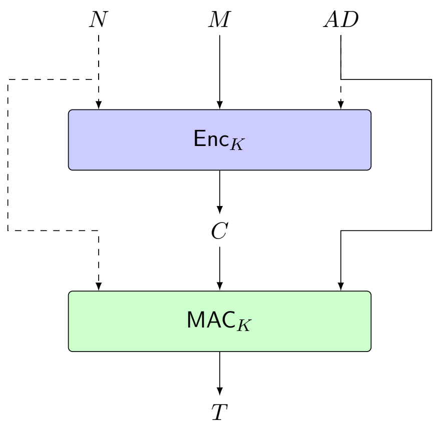
$$\mathsf{AE}_{\mathsf{enc}}(K, N, AD, M) = (C, T).$$

The decryption and verification algorithm $\mathsf{AE}_{\mathsf{dec}}$ takes as input the secret key K, nonce N, associated data AD, ciphertext C and tag T, and outputs the plaintext M of the same length as C only if the verification of tag is correct, and $\perp$ (error symbol) if the tag verification fails. More formally,

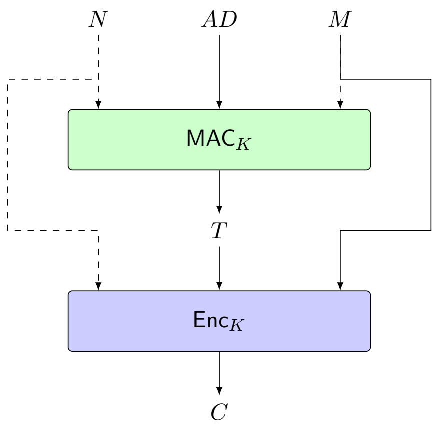
$$\mathsf{AE}_{\mathsf{dec}}(K, N, AD, C, T) \in \{M, \perp\}.$$

2.3.5.1 Constructions of AEAD

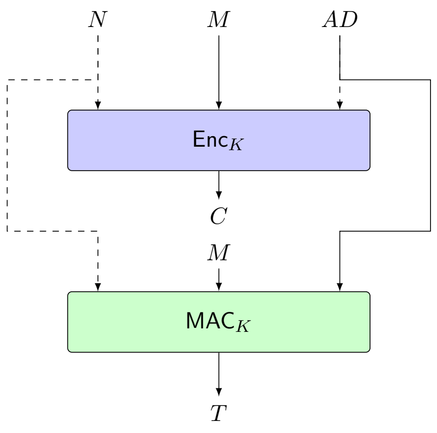
Figure 2.17 shows the three main approaches to construct an AEAD algorithm. They are 1) *Encrypt-then-MAC*, 2) *MAC-then-Encrypt* and 3) *Encrypt-and-MAC*.



(a) Encrypt-then-MAC



(b) MAC-then-Encrypt



(c) Encrypt-and-MAC

Figure 2.17: Generic constructions of AEAD. Dotted line denotes that the input vary with respect to specific instantiations of encryption and MAC algorithms.

Encrypt-then-MAC. The sender initially computes the ciphertext C and then uses it as an input to MAC algorithm for the tag generation (Figure 2.17 (a)). For verification, the receiver computes the tag first, and if it matches with the received tag, then only he decrypts C.

MAC-then-Encrypt. In this case, tag T is computed first. Next, T and the plaintext M are used as input to the encryption algorithm for generating C (Figure 2.17 (b)). For verification, the receiver has to decrypt C to obtain M. The plaintext is then fed to the MAC algorithm which outputs tag T'. If T' equals T, then only verification is successful.

Encrypt-and-MAC. In this approach, encryption and MAC computations are done in parallel. However, similar to the MAC-then-Encrypt approach, a receiver has to perform decryption first (Figure 2.17 (c)).

2.3.5.2 Mode of operations for AEAD

In the previous section, we have seen three different approaches of AEAD construction. However, both encryption and MAC procedures require a mode of operation to handle long messages. We now explain different AEAD modes based on block ciphers and sponge construction.

Block cipher based AEAD. Galois Counter Mode (GCM) [96], OCB [114] and OCB3 [88] are widely adopted AEAD modes of operation based on Encrypt-then-MAC paradigm. Roughly speaking, all three modes adopt the counter mode of operation for encryption. For OCB, the input and output of the block cipher is randomized by XORing a secret mask. On the other hand, OCB3 uses a tweakable block cipher where the tweak value acts as a counter. For OCB and OCB3, authentication is achieved by an additional call of primitive after the processing of associated data and plaintext, while GCM authentication is based on the evaluation of a polynomial function in a finite field. The exact structures of these modes (for simplicity, only two full blocks are considered) are shown in Figures 2.18-2.20.

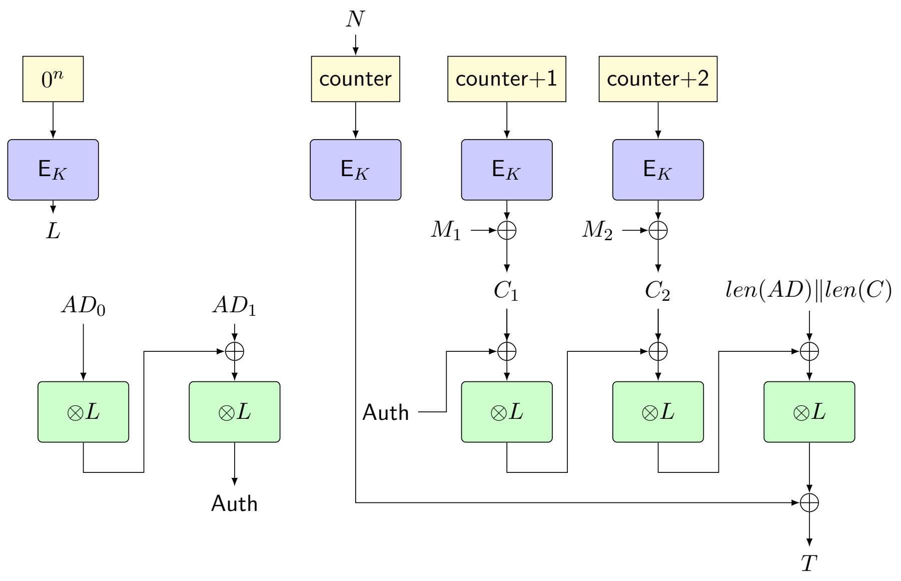


Figure 2.18: GCM mode where $\otimes$ denotes the finite field multiplication in $\mathbb{F}_{2^n}$

Remark 2.4. The choice of secret masks (Figure 2.19) and tweaks (Figure 2.20) are crucial to prevent forgery attacks and distinguishing different processing phases. In Chapter 10, we

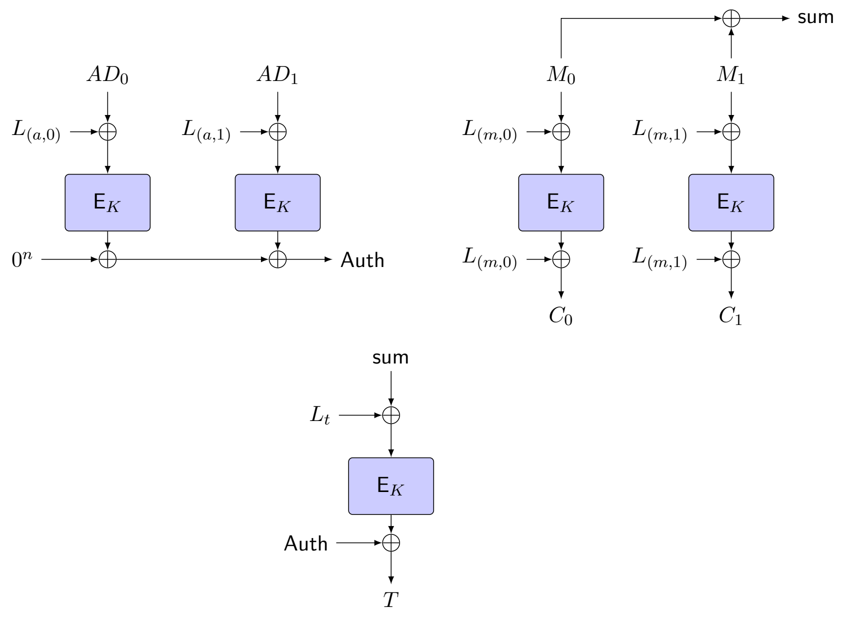


Figure 2.19: OCB mode where $L_{(a,i)}$, $L_{(m,i)}$ and L_t are secret masks obtained from N and K

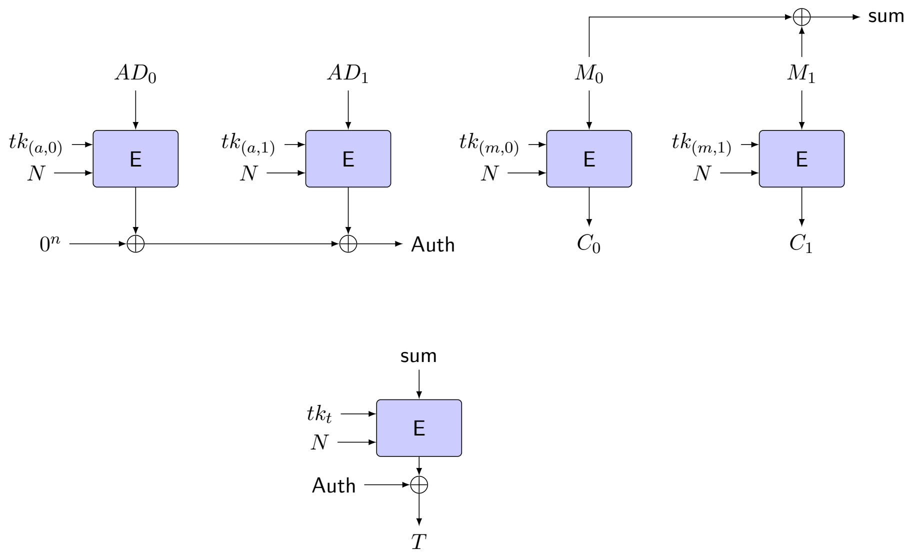


Figure 2.20: OCB3 mode where $tk_{(a,i)}$, $tk_{(m,i)}$ and tk_t are tweaks

present practial forgery attacks on some AEAD modes which exploit weaknesses in secret masks.

Sponge based AEAD. A sponge based AEAD is a permutation based sequential mode of operation. Let P be a b-bit permutation with $b = r + c$ where r and c denote the rate and capacity part of the state, respectively. The entire algorithm is divided into 4 phases which are illustrated in Figure 2.21 and described below.

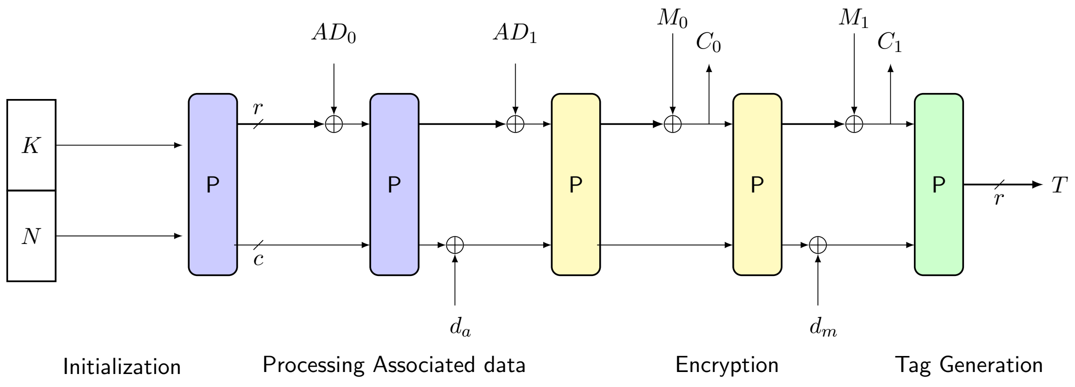


Figure 2.21: Schematic of AEAD using sponge where d_a and d_m are the domain separators. For simplicity, we have considered the processing of two complete AD and M blocks that are obtained after padding. Furthermore, tag size equals r.

1. *Initialization*: The state is first loaded with K and N, and then P is called once. The goal is to have a random looking initial state after the single call of P.
2. *Processing Associated data*: Each r-bit AD block is XORed to the rate and then P is applied. During the processing of last AD block, a domain separator d_a is XORed to the capacity to indicate that the current block is the last block of associated data.
3. *Encryption*: Each r-bit M block is XORed to the rate which gives the ciphertext. Next, the ciphertext block is fed to the state and then P is applied. While processing the last block of M, a domain separator d_m is XORed to the capacity to indicate that the current block is the last message block.
4. *Tag Generation/Finalization*: The tag is extracted from the rate part of the state r-bits at a time.

Figure 2.21 describes sponge based AEAD scheme at a very high level. There exists many variants of the same structure with tweaks in the above mentioned 4 phases and domain separators. A majority of NIST LWC round 2 candidates are based on the sponge construction [5]. Chapters 6 and 7 present two such variants.

2.4 Generic Cryptanalysis Techniques

Cryptanalysis is the art of finding hidden aspects of a cryptographic primitive or a cryptographic protocol that can be exploited by an attacker to learn something about the key or the plaintext. It is a broad and never-ending field as attacks often get better with time. Cryptanalysis is categorized into two major branches, namely *mathematical cryptanalysis* and *implementation attacks*. The former targets the mathematical structure while the later exploits side channel information such as power consumption or execution time to analyze a cryptosystem. In both cases, to launch a attack, an attacker requires *data*, *time* and *memory* storage which are called *attack complexity metrics*. The better the attack complextity metrics the better is the attack.

In the following, we first explain different adversarial models[5] and attack goals. Next, we discuss generic attack techniques which are essential from designer's perspective.

2.4.1 Adversarial models and goals

An *adversarial model* is the set of rules followed by an adversary for an attack. There are six major adversarial models based on the type of data available to an adversary.

1. *Ciphertext-only:* Only ciphertexts.
2. *Known-plaintext:* The plaintexts and their corresponding ciphertexts.
3. *Chosen-ciphertext:* Adversary chooses a ciphertext C, queries it to the decryption oracle and obtains the corresponding plaintext P.
4. *Chosen-plaintext:* Adversary chooses a plaintext P, queries it to the encryption oracle and obtains the corresponding ciphertext C.
5. *Adaptive-chosen-ciphertext:* This is similar to chosen-ciphertext model. The only difference is that after observing N plaintext and ciphertext pairs (M_i, C_i) $i = 1$ to N, the adversary adaptively chooses the next ciphertext based on the previous N pairs.
6. *Adaptive-chosen-plaintext:* This is a dual of adaptive-chosen-ciphertext. In this scenario, the adversary adaptively chooses the next plaintext based on the previous N pairs.

Note that for ciphertext-only and known-plaintext models, there is no interaction with the oracles. In addition, for each of these models it is necessary to define in which setting those data are obtained.

- **Single-key setting.** Adversary obtains the data corresponding to a fixed secret key.
- **Related-key setting.** Adversary gets the victim or oracle to encrypt/decrypt with a key that is related in a chosen way to the original key.
- **Nonce-misuse setting.** Adversary is allowed to repeat the nonce for both encryption and decryption queries.

[5]Our focus here is on mathematical cryptanalysis

- **Nonce-respecting setting.** Adversary is not allowed to repeat the nonce for an encryption query. However, for a decryption query nonce can be repeated. This is the case of forgery attacks when adversary tries to pass the verification by repeating nonce and changing ciphertext and/or associated data.

Now, once the adversary has enough data corresponding to a particular setting and specific model, he tries to achieve either of the following goals.

1. *Distinguishing attack:* In this scenario, the adversary tries to distinguish whether the data he obtained is from the real world (i.e., the actual encryption/decryption oracle) or the ideal world. In an ideal world, an idealized oracle outputs a fixed length string by picking it randomly from the uniform distribution.
2. Key recovery attack or breaking the confidentiality, integrity or authenticity.

2.4.2 Differential cryptanalysis

Differential cryptanalysis is one of the most powerful cryptanalytic techniques against symmetric key primitives. It was proposed by Biham and Shamir [38] to cryptanalyze the block cipher DES, and subsequently other ciphers. The attack exploits the fact that a single round of a primitive is usually weaker than multiple rounds.

2.4.2.1 Basic idea and definitions

Let $\mathcal{RF} : \{0,1\}^n \to \{0,1\}^n$ be an n to n bit vectorial boolean function and $X \sim U(\{0,1\}^n)$. The idea is to find a pair of input and output difference α and $\beta \in \{0,1\}^n$ with $\alpha \neq 0$ for which the probability $\mathsf{Pr}(\mathcal{RF}(X) + \mathcal{RF}(X + \alpha) = \beta)$ is maximum. For a random $\mathcal{RF}$ the value is close to 2^{-n}. In case $\mathcal{RF}$ is a cryptographic primitive, we aim to find (α, β) for which it is greater than 2^{-n}. The pair can then be used as a distinguisher or for key recovery attack as described later. This value is often called *differential probability* and we formally define it in Definition 2.1.

Definition 2.1 (Differential Probability (DP) [38, 107])**.** For a given α and β, the differential probability is given by

$$\mathsf{Pr}(\mathcal{RF}(X) + \mathcal{RF}(X + \alpha) = \beta) = \frac{|\{x \mid \ \mathcal{RF}(x) + \mathcal{RF}(x + \alpha) = \beta\}|}{2^n}.$$

Moreover, we call (α, β) a *differential.*

Typically, a cryptographic primitive is an iterative structure and the final output is obtained after applying the round function $\mathcal{RF}$ multiple times (say r). Accordingly, we have a sequence of random variables $\Delta Y_0, \Delta Y_1, \cdots, \Delta Y_r$, which we call an *$r$-round differential characteristic* (Figure 2.22). The differential characteristic satisfies

$$\Delta Y_i = \alpha_i = x_i + x_i', \text{ for } 0 \leq i \leq r \text{ and}$$
$$\mathsf{Pr}(\Delta Y_i = \alpha_i, \Delta Y_{i+1} = \alpha_{i+1}) > 0 \text{ for } 0 \leq i \leq r-1.$$

Computing the exact probability of an r-round differential characteristic is infeasible because of the exponential search space in n. A common approach is to exploit the properties of the round function, compute the probability of a single round differential, assume the independence of rounds, and then take the product of probabilities (Definition 2.2). By independence of rounds, we mean that the differential probability of the output difference at the $(i+1)$-th round depends only on the output difference at round i. Note that the assumption of independence of rounds is not true in general (as we see later), however it provides a good approximation in practice.

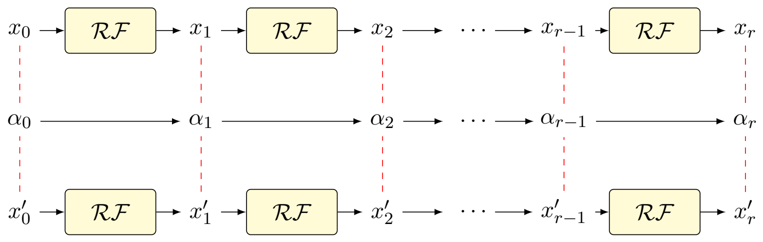


Figure 2.22: r-round differential characteristic

Definition 2.2 (Differential Characteristic Probability (DCP))**.** Let $r > 0$ and $\Delta Y_0 = \alpha_0, \Delta Y_1 = \alpha_1, \cdots, \Delta Y_r = \alpha_r$ be an r-round differential characteristic. Assuming that all rounds are independent, the *r-round differential characteristic probability* is given by

$$\mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_1 = \alpha_1, \cdots, \Delta Y_r = \alpha_r) = \prod_{i=0}^{r-1} \mathsf{Pr}(\Delta Y_i = \alpha_i, \Delta Y_{i+1} = \alpha_{i+1}).$$

As an adversary, one is only interested in the differential (α_0, α_r) and not the intermediate differences. This is because there might be multiple paths with the same input and output difference. Thus, the probability of an *r-round differential* (α_0, α_r) is always at least the probability of a single characteristic. We introduce the following definition to reflect this fact.

Definition 2.3 (r-round Differential Probability)**.** For an r round differential (α_0, α_r), its probability is given by

$$\mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r) = \sum_{\alpha_1, \ldots, \alpha_{r-1}} \mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_1 = \alpha_1, \cdots, \Delta Y_r = \alpha_r).$$

Remark 2.5. Definition 2.1 is a special case of Definition 2.3 with $r = 1$.

The phenomenon in Definition 2.3 is often termed *differential effect*. For simplicity, let $w = \lceil -\log_2(\mathsf{DCP}(\cdot)) \rceil$ denote the weight of a differential characteristic. Then, an alternate expression of an r-round differential is given by

$$\mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r) = \sum_{i \neq 0} s_i 2^{-i} \tag{2.1}$$

where s_i is the number of r-round differential characteristics with $w = i$.

2.4.2.2 Effect of round keys

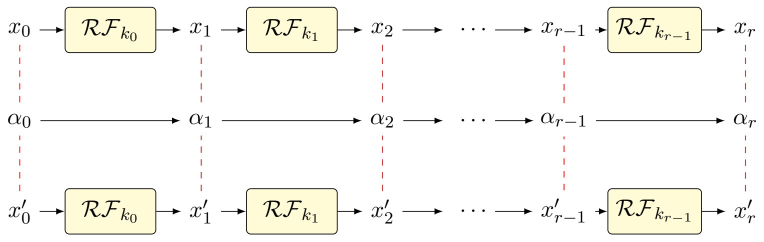


Figure 2.23: r-round differential characteristic with keyed round function

In the previous definitions, we have ignored the effect of round key additions. However, the round function $\mathcal{RF}$ is usually parameterized by a round key (e.g., block cipher). We denote it by $\mathcal{RF}_{k_i}$ where k_i is the i-th round key and K_i is the random variable corresponding to k_i.

Now, consider an r-round differential as shown in Figure 2.23. The main problem here is that we need to compute the value of

$$\mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r \mid K_0 = k_0, \cdots, K_{r-1} = k_{r-1}) \tag{2.2}$$

without the knowledge of secret key. Thus, in most of the cases, we are only able to compute the value of $\mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r)$ by assuming that

1. $K_i \sim U(\{0,1\}^n)$ for all i.
2. For all $i \neq j$, K_i and K_j are independent random variables.

Remark 2.6. Both the above assumptions are not true in general as the round keys are derived from the master key which has fixed size.

Furthermore, we have to assume that $\mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r)$ is roughly same for almost all keys. This additional condition is referred to as *the hypothesis of stochastic equivalence*.

Definition 2.4 (Hypothesis of stochastic equivalence [90])**.** For an r-round differential (α_0, α_r)

$$\mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r \mid K_0 = k_0, \cdots, K_{r-1} = k_{r-1}) \approx \mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r)$$

for almost all round keys $k_0, \cdots, k_{r-1}$.

In practice, the above hypothesis may not hold as pointed out by Canteaut [45]. This means a differential may have low probability on average but for some keys[6] its probability may be high.

2.4.2.3 Expected differential probabilities and Markov ciphers

Let E_k be a keyed cipher with r rounds, i.e. $\mathsf{E}_k(\cdot) = \mathcal{RF}_{k_{r-1}} \circ \cdots \circ \mathcal{RF}_{k_0}(\cdot)$.

Definition 2.5 (Expected Differential Probability (EDP) [90])**.** For an r-round differential (α_0, α_r), the *expected differential probability* is given by

$$\mathsf{EDP}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r) = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \mathsf{Pr}(\mathsf{E}_k(X) + \mathsf{E}_k(X + \alpha_0) = \alpha_r).$$

Definition 2.6 (Markov cipher (cf. Page 6 [90]))**.** We say E_k is a *Markov cipher* if for all $0 \leq i \leq r-1$,

$$\mathsf{Pr}(\mathcal{RF}_{k_i}(x_i) + \mathcal{RF}_{k_i}(x_i + \alpha_i) = \alpha_{i+1} \mid X_i = x_i) = \mathsf{Pr}(\mathcal{RF}_{k_i}(x_i) + \mathcal{RF}_{k_i}(x_i + \alpha_i) = \alpha_{i+1})$$

for all choices of x_i and uniformly random chosen k_i.

For a Markov cipher with all independent round keys, the EDP can be estimated as follows

$$\mathsf{EDP}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r) \approx \mathsf{Pr}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r).$$

From Definition 2.5, it is clear that if one needs to compute EDP for a given (α_0, α_r), the number of encryption queries is of order $\mathcal{O}(2^{n+|K|})$. By Markov assumption, this is reduced to $\mathcal{O}(2^n)$. The complexity is for a given (α_0, α_r), and is still not practical. Thus, an adversary tries to find a differential characteristic with maximum probability rather than a differential. The former is easier to compute, thanks to automated tools such as CryptoSMT and MILP solvers [122, 2].

Let $\mathcal{V}_r$ denote the set of all r-round differential characteristics. Then, the *Maximum Expected Differential Characteristic Probability* (MEDCP) is defined as follows.

[6]considered as weak keys

Definition 2.7 (MEDCP [83, 58]). For a r-round Markov cipher E_k^r, its MEDCP is given by

$$\mathsf{MEDCP}(\mathsf{E}_k^r) = \max_{\alpha_0,\cdots,\alpha_r \in \mathcal{V}_r} \prod_{i=0}^{r-1} \mathsf{Pr}(\Delta Y_i = \alpha_i, \Delta Y_{i+1} = \alpha_{i+1})$$

We call a differential characteristic with probability equal to $\mathsf{MEDCP}(\mathsf{E}_k^r)$ an *optimal r-round differential characteristic.*

2.4.2.4 Differential distinguishers and key recovery attack

Consider a typical example of a block cipher with r rounds as shown in Figure 2.24. The differential attack is a chosen plaintext/chosen-ciphertext attack. Thus, the adversary could choose either of encryption or decryption oracle. We focus on the chosen-plaintext scenario here. Let (α_0, α_{r-1}) be an $(r-1)$-round differential with probability $p > 2^{-n}$.

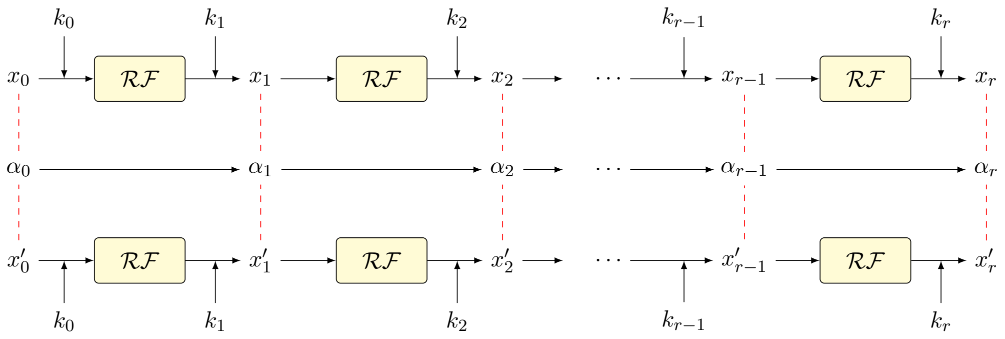


Figure 2.24: Difference propagation for a typical block cipher

Distinguishing attack. The distinguishing attack works as follows.

Step 1 Choose a random x_0. Query x_0 and $x'_0 = x_0 + \alpha_0$ to the oracle, and obtain x_{r-1} and x'_{r-1}.

Step 2 Compute $x_{r-1} + x'_{r-1}$.

Step 3 Repeat Step 1 and Step 2.

If the oracle is an encryption oracle, then after N trials, we expect that Np values in Step 2 satisfy $x_{r-1} + x'_{r-1} = \alpha_{r-1}$. For an ideal oracle, the output difference looks random. Thus, on average $N = \frac{1}{p}$ plaintext-ciphertext pairs are needed for the distinguishing attack.

Key recovery attack. We now show how to exploit an $(r-1)$-round differential distinguisher for an r round key recovery attack. Assume that the attacker obtains N plaintext/ciphertext pairs (P_i, C_i) and $(P_i + \alpha_0, C'_i)$ for $i = 1, \ldots, N$. The attack then proceeds as follows.

Step 1 Initialize an array of counter CNT with $\mathsf{CNT}[i] = 0$ for $i = 0, \ldots, 2^{\kappa} - 1$ where $|k_r| = \kappa'$.

Step 2 For each of $2^{\kappa'}$ guesses of k_r and for each i, decrypt one round, and compute $x_{r-1} = \mathcal{RF}^{-1}(C_i + k_r)$, $x'_{r-1} = \mathcal{RF}^{-1}(C'_i + k_r)$. If $x_{r-1} + x'_{r-1}$ equals α_{r-1}, increment the corresponding key counter by 1.

Note that there might be false positives, i.e., wrong keys which satisfy $x_{r-1} + x'_{r-1} = \alpha_{r-1}$. Let p' be the probability of false positives and assume that $p' \ll p$. Since the differential probability is p, we repeat the experiment $N = \frac{c}{p}$ times for some constant c. Thus, one of the counter values will be significantly higher (in fact, equal to c) than others. The corresponding key is then considered as a right key candidate for k_r.

Attack complexities. In Step 2, we do not guess all last round keys in an actual attack. We exploit the properties of $\mathcal{RF}$ such as the sboxes which are affected by the difference (α_{r-1}, α_r). The k_r values are then guessed accordingly. Let $N = \frac{c}{p}$ be the number of plaintext-ciphertext pairs and l be the number of such keys. Then, the attack time complexity is $N \times 2^{l+1}$ 1-round decryptions. Note that this highly depends on $\mathcal{RF}$.

2.4.3 Linear cryptanalysis

Linear cryptanalysis was introduced by Matsui and Atsuhiro [95] to cryptanalyze the FEAL cipher. Contrary to the differential attack, it is a known-plaintext attack where an adversary tries to approximate the output of the cipher with a linear boolean function of the input.

2.4.3.1 Basic idea and definitions

Let $\mathcal{RF} : \{0,1\}^n \to \{0,1\}^n$ be an n to n bit vectorial boolean function and $X \sim \mathsf{Unif}(\{0,1\}^n)$. The idea is to find a pair of input and output masks α and $\beta \in \{0,1\}^n$ with $\alpha \neq 0$ for which the bias $\epsilon_{\alpha,\beta}$ in the equation

$$\mathsf{Pr}(< X, \alpha > = < \mathcal{RF}(X), \beta >) = \frac{1}{2} + \epsilon_{(\alpha,\beta)}$$

is maximum. Let $\hat{\mathcal{RF}}(\alpha, \beta) = \sum_{x \in \{0,1\}^n} (-1)^{<x,\alpha> + <\mathcal{RF}(x),\beta>}$ denote the Fourier coefficient of $\mathcal{RF}$ with respect to α and β. Then, an alternate form of computing bias in terms of *square correlation* is defined in Definition 2.8.

Definition 2.8 (Square correlation (SC) [105]). For a given α and β, the square correlation is given by

$$\mathsf{SC}(\alpha, \beta) = \Big(\frac{\hat{\mathcal{RF}}(\alpha, \beta)}{2^n}\Big)^2.$$

Moreover, we call (α, β) a *linear hull*.

Analogous to an r-round differential characteristic, the linear masks form a sequence of boolean random variables $\Delta Y_0, \ldots, \Delta Y_r$ where $\Delta Y_i =< x_i, \alpha_i >$ for $i = 0, \ldots, r$. We call it an *r-round linear characteristic*. Accordingly, we have the following definitions similar to Definitions 2.2-2.7.

Definition 2.9 (Linear Characteristic Square Correlation (LCSC))**.** Let $r > 0$ and $\Delta Y_0 = \alpha_0, \Delta Y_1 = \alpha_1, \cdots, \Delta Y_r = \alpha_r$ be an r-round linear characteristic. Assuming that ΔY_i's are independent for all i, then the *r-round linear characteristic square correlation* is given by

$$\mathsf{SC}(\Delta Y_0 = \alpha_0, \Delta Y_1 = \alpha_1, \cdots, \Delta Y_r = \alpha_r) = \prod_{i=0}^{r-1} \mathsf{SC}(\Delta Y_i = \alpha_i, \Delta Y_{i+1} = \alpha_{i+1}).$$

Definition 2.10 (r-round Linear Hull Square Correlation (LHSC))**.** For an r-round linear hull (α_0, α_r), its square correlation is given by

$$\mathsf{SC}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r) = \sum_{\alpha_1, \ldots, \alpha_{r-1}} \mathsf{SC}(\Delta Y_0 = \alpha_0, \Delta Y_1 = \alpha_1, \cdots, \Delta Y_r = \alpha_r).$$

The phenomenon in Definition 2.10 is often termed *linear effect*. For simplicity, let $w = \lceil -\log_2(\mathsf{LCSC}(\cdot)) \rceil$ denote the weight of a linear characteristic. Then, an alternate expression of an r-round linear hull is given by

$$\mathsf{SC}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r) = \sum_{i \neq 0} s_i 2^{-i} \tag{2.3}$$

where s_i is the number of r-round linear characteristics with $w = i$.

Definition 2.11 (Expected Linear Hull Square Correlation (ELHSC))**.** For an r-round linear hull (α_0, α_r), the *expected linear hull square correlation* is given by

$$\mathsf{ELHSC}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r) = \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} \mathsf{SC}(\Delta Y_0 = \alpha_0, \Delta Y_r = \alpha_r).$$

Definition 2.12 (Maximum Expected Linear Characteristic Square Correlation (MELCSC))**.** Let E_k^r be a Markov cipher with r rounds and let $\mathcal{L}_r$ be the set of all r-round linear characteristics. Then, MELCSC is given by

$$\mathsf{MELCSC}(\mathsf{E}_k^r) = \max_{\alpha_0, \cdots, \alpha_r \in \mathcal{L}_r} \prod_{i=0}^{r-1} \mathsf{SC}(\Delta Y_i = \alpha_i, \Delta Y_{i+1} = \alpha_{i+1})$$

We call a linear characteristic with probability equal to $\mathsf{MELCSC}(\mathsf{E}_k^r)$ an *optimal r-round linear characteristic*.

2.4.3.2 Linear hull as a distinguisher

Let (α_0, α_r) be an r-round linear hull with square correlation greater than 2^{-n+1}. The distinguishing attack work as follows.

Step 1 Initialize two counters $\mathsf{CNT}_0 = 0$ and $\mathsf{CNT}_1 = 0$.

Step 2 Query x_0 and x_r.

Step 3 Compute $< x_0, \alpha_0 > + < x_r, \alpha_r >$. If $< x_0, \alpha_0 > + < x_r, \alpha_r >$ equals zero, increment CNT_0, else increment CNT_1.

Step 4 Repeat Step 1 and Step 2.

For an ideal oracle, CNT_0 and CNT_1 are roughly equal to $\frac{N}{2}$ after N trials. However, CNT_0 is biased in case of an encryption oracle, i.e., if the square correlation is p then $\mathsf{CNT}_0 = \frac{N}{2} + \frac{N}{\sqrt{p}}$.

2.4.4 High order differential attacks

Recall Definition 2.1 of differential probability, i.e., $\mathsf{Pr}(\mathcal{RF}(X) + \mathcal{RF}(X + \alpha) = \beta)$. Alternatively, it is the probability that first order derivative of $\mathcal{RF}$ at α equals β when X follows a uniform distribution. The idea was generalized to *higher order derivatives* by Lai [89] and later extended to integral attacks [85] and division propery based attacks [126]. The high level idea of these attacks can be described as follows. Let f be a boolean function with algebraic degree d, then its $(d+1)$-th order derivative is constant. In particular, evaluating f on 2^{d+1} points and then summing the output, the final sum is zero with probability 1.

2.4.4.1 Division property

We present a toy example (Table 2.1) before defining it formally in Definitions 2.13 and 2.14. Let $\pi_u(x) : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2$ be defined as $\pi_u(x) = \prod_{i=0}^{n-1} x_i^{u_i}$ where $u, x \in \mathbb{F}_2^n$ and w_u denotes the hamming weight of vector u. For example: $u = (101)$, then $\pi_u(x) = x_2^1 x_1^0 x_0^1 = x_2 x_0$. If $x = (110)$ then $\pi_u(x) = 0$. We observe that $\forall$ u satisfying

- $w_u < 3 \implies \bigoplus_{x \in \mathbb{F}_2^3} \pi_u(x) = 0$, and $w_u = 3 \implies \bigoplus_{x \in \mathbb{F}_2^3} \pi_u(x)$ becomes unknown,

- $w_u < 2 \implies \bigoplus_{x \in \mathbb{F}_2^3} \pi_u(x) = 0$, and $w_u \geq 2 \implies \bigoplus_{x \in \mathbb{F}_2^3} \pi_u(x)$ becomes unknown.

Definition 2.13 (Word based division property [126]). Let $\mathbb{X} \subseteq \mathbb{F}_2^n$, $0 \leq k \leq n$, we say that $\mathbb{X}$ has the *division property* $\mathcal{D}_k^n$ if

$$\bigoplus_{x \in \mathbb{X}} \pi_u(x) = 0, \text{ for all } u \in \mathbb{F}_2^n \text{ s.t } w_u < k.$$

Table 2.1: Toy example for the division property

$u\backslash x$	000	001	010	011	100	101	110	111	$\bigoplus_x \pi_u(x)$
000	1	1	1	1	1	1	1	1	0
001	0	1	0	1	0	1	0	1	0
010	0	0	1	1	0	0	1	1	0
011	0	0	0	1	0	0	0	1	0
100	0	0	0	0	1	1	1	1	0
101	0	0	0	0	0	1	0	1	0
110	0	0	0	0	0	0	1	1	0
111	0	0	0	0	0	0	0	1	1

Moreover, $\mathbb{X}$ is called as a *multiset*.

Definition 2.14 (Bit based division property [128])**.** Let $\mathbb{X}, \mathbb{W} \subseteq \mathbb{F}_2^n$. The multiset $\mathbb{X}$ has the division property $\mathcal{D}_{\mathbb{W}}^{1,n}$ if it fulfills the following conditions:

$$\bigoplus_{x \in \mathbb{X}} \pi_u(x) = \begin{cases} \textit{unknown} & \textit{if there exists } w \in \mathbb{W} \textit{ s.t } u \succeq w, \\ 0 & \textit{otherwise} \end{cases}$$

where $u, w, x \in \mathbb{F}_2^n$ and we denote $u \succeq w$ if $u_i \geq w_i$ for all i.

Table 2.2: Bit based division property propagation rules

	Division property	
Operation	Input multiset $\mathbb{X}$	Output multiset $\mathbb{Y}$
COPY: $x \to (y_0, y_1)$	$\mathcal{D}^{1,1}_{\{(0)\}}$	$\mathcal{D}^{1,2}_{\{(0,0)\}}$
	$\mathcal{D}^{1,1}_{\{(1)\}}$	$\mathcal{D}^{1,2}_{\{(0,1),(1,0)\}}$
XOR: $x_0 \oplus x_1 \to y$	$\mathcal{D}^{1,2}_{\{(0,0)\}}$	$\mathcal{D}^{1,1}_{\{(0)\}}$
	$\mathcal{D}^{1,2}_{\{(0,1)\}}$	$\mathcal{D}^{1,1}_{\{(1)\}}$
	$\mathcal{D}^{1,2}_{\{(1,0)\}}$	$\mathcal{D}^{1,1}_{\{(1)\}}$
AND: $x_0 x_1 \to y$	$\mathcal{D}^{1,2}_{\{(0,0)\}}$	$\mathcal{D}^{1,1}_{\{(0)\}}$
	$\mathcal{D}^{1,2}_{\{(0,1)\}}$	$\mathcal{D}^{1,1}_{\{(1)\}}$
	$\mathcal{D}^{1,2}_{\{(1,0)\}}$	$\mathcal{D}^{1,1}_{\{(1)\}}$
	$\mathcal{D}^{1,2}_{\{(1,1)\}}$	$\mathcal{D}^{1,1}_{\{(1)\}}$

Definitions 2.13 and 2.14 consider the propagation rules for a single round. Thus, similarly to an r-round differential/linear characteristic we define an *r-round division trail* as follows.

Definition 2.15 (r-round division trail [134])**.** Let $\{v\} \overset{def}{=} \mathbb{W}_0 \to \mathbb{W}_1 \to \cdots \to \mathbb{W}_r$ be an r-round division property propagation where $\mathbb{W}_i \subseteq \mathbb{F}_2^n$ for $0 \leq i \leq r$. We call $(w_0, w_1, \ldots, w_r) \in$

$\mathbb{W}_0 \times \mathbb{W}_1 \times \ldots \times \mathbb{W}_r$ an r-round division trail if w_{i-1} can propagate to w_i by division property propagation rules for all $i \in \{1, 2, \ldots, r\}$ (Table 2.2).

Example 2.1. Let $f : \mathbb{F}_2^3 \to \mathbb{F}_2^3$ given by $f(x_0, x_1, x_2) = (x_0x_1 \oplus x_2, x_0, x_1)$ and $\mathbb{W}_0 = \{(1, 1, 0)\}$, then $\mathbb{W}_1 = \{(1, 0, 0), (1, 0, 1), (0, 1, 1)\}$ and $\mathbb{W}_2 = \{(1, 0, 0), (0, 1, 0), (1, 0, 1)\}$.

MILP models for division property. Finding r-round division trails was infeasible due to large number of vectors in $\mathbb{W}_i$ until Xiang *et al.* [134] showed how to model the division property propagation rules using MILP. As a result, they found the best known integral disinguishers for block ciphers Simon, Simeck, Present, Rectangle and Twine. We now explain how to model COPY, XOR and AND using MILP. We use M to denote the MILP model.

- *MILP model for COPY.* Let the division trail through a copy operation be denoted by $a \to (b_1, b_2, \ldots, b_m)$, then the following inequalities are used to model such propagation:

$$M.var \leftarrow a, b_1, b_2, \ldots, b_m \text{ as binary.}$$
$$M.con \leftarrow a = b_1 + b_2 + \ldots + b_m.$$

- *MILP model for XOR.* Let $(a_1, a_2, \cdots, a_m) \to b$ denote the division trail of XOR, then the following inequalities are sufficient to describe the propagation of the division property:

$$M.var \leftarrow a_1, a_2, \cdots a_m, b \text{ as binary.}$$
$$M.con \leftarrow a_1 + a_2 + \cdots + a_m = b.$$

- *MILP model for AND.* Let $(a_1, a_2, \cdots, a_m) \to b$ denote the division trail for AND, then the following inequalities are used to describe the propagation:

$$M.var \leftarrow a_1, a_2, \cdots a_m, b \text{ as binary.}$$
$$M.con \leftarrow b \geq a_i \text{ for } i = 1, 2, \cdots, m.$$

2.4.4.2 Cube attacks

The cube attack proposed in [59, 129] is a powerful cryptanalytic technique against stream ciphers. The idea is to analyze the Algebraic Normal Form (ANF) of summation of the first keystream bit corresponding to a set of public input variables. For example, let $f : \mathbb{F}_2^5 \to \mathbb{F}_2$ given by

$$f(k_0, k_1, k_2, v_0, v_1) = v_0v_1k_0 \oplus v_0v_1k_2 \oplus v_0v_1 \oplus k_0k_1 \oplus v_1k_2 \oplus k_2 \oplus 1$$

$$\implies f(k_0, k_1, k_2, v_0, v_1) = v_0v_1(k_0 \oplus k_2 \oplus 1) \oplus k_0k_1 \oplus v_1k_2 \oplus k_2 \oplus 1$$

where k_0, k_1, k_2 are secret variables while v_0 and v_1 are public variables. Summing f over all possible choices of v_0, v_1 gives $f(k_0, k_1, k_2, 0, 0) \oplus f(k_0, k_1, k_2, 0, 1) \oplus f(k_0, k_1, k_2, 1, 0) \oplus$

$f(k_0, k_1, k_2, 1, 1) = k_0 \oplus k_2 \oplus 1$, which is a linear relation of the two key bits k_0 and k_2. In cube attacks, such relationships are exploited to recover secret bits.

Mathematical description. Let the stream cipher take an n-bit secret key $k = (k_0, \cdots, k_{n-1})$ and an m-bit $IV = (v_0, v_1, \cdots, v_{m-1})$, then the first keystream bit is given by the polynomial $f(k, v)$ which operates on $n + m$ bits to output 1 bit and can be represented as:

$$f(k, v) = t_I \cdot p(k, v) \oplus q(k, v)$$

where $I = \{i_1, i_2, \cdots, i_{|I|}\} \subseteq \{0, 1, \cdots, m - 1\}$, $t_I = v_{i_1} v_{i_2} \cdots v_{i_{|I|}}$, $p(k, v)$ is a polynomial that does not contain any of the variables $(v_{i_1}, v_{i_2}, \cdots, v_{i_{|I|}})$, and $q(k, v)$ is independent of at least one variable from $(v_{i_1}, v_{i_2}, \cdots, v_{i_{|I|}})$.

We denote a cube indices set by I and the corresponding cube by C_I where C_I is the set of all the possible $2^{|I|}$ values of $(v_{i_1}, v_{i_2}, \cdots, v_{i_{|I|}})$. The remaining input $n + m - |I|$ variables to set to some constant values and the summation of $f(k, v)$ over all values of the cube C_I is then given by

$$\bigoplus_{C_I} f(k, v) = (\bigoplus_{C_I} t_I \cdot p(k, v)) \bigoplus (\bigoplus_{C_I} q(k, v)).$$

Since such summation reduces t_I to 1 because the set C_I has only one possibility where all the $|I|$ variables are equal to 1, and $q(k, v)$ vanishes because it misses at least one variable from the cube variables, then the above equation denotes the *superpoly* which is given by

$$\textit{superpoly} =: \bigoplus_{C_I} f(k, v) = p(k, v).$$

If the ANF of the superpoly is simple enough, then an attacker can query the encryption oracle with the chosen cube C_I. Hence, the returned first keystream bits are summed to evaluate the right-hand side of the superpoly and accordingly, secret variables can be recovered by solving a system of equations. For more detailed examples, please see Section 3.7 in [48].

Cube attacks and division property. In [127], Todo *et al.* proposed a method to apply cube attacks on stream ciphers using the division property. Since the cube attack is a higher-order differential attack and the division property is a technique to find higher-order differential trails, the division property can then be used to analyze the ANF of the superpoly by analyzing division trails corresponding to a given cube.

2.4.5 Meet-in-the-middle attacks

The meet-in-the-the middle (MitM) attack is a generic Time-Memory-Data (TMD) trade-off attack proposed by Diffie and Hellman [57]. The idea is to decompose an encryption algorithm E as a composition of two subciphers E_f and E_b (Figure 2.25) such that $C = \mathsf{E}(k, P) = \mathsf{E}_b(k_b, \mathsf{E}_f(k_f, P))$.

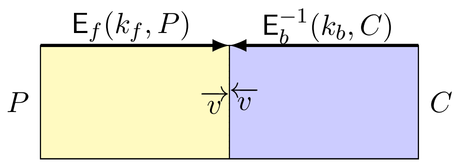


Figure 2.25: MitM attack

The steps of a standard MitM attack are divided into two phases.

1. **MitM phase.** For all possible values of k_f, compute $\overrightarrow{v} = \mathsf{E}_f(k_f, P)$ and store $(k_f, \overrightarrow{v})$ in data structure DS. Compute $\overleftarrow{v} = \mathsf{E}_b^{-1}(k_b, C)$ and check if $\overleftarrow{v} \in \mathsf{DS}$. If so, the value of k corresponding to the pair (k_f, k_b) is one of the key candidates.

2. **Brute force phase.** If the number of keys obtained from MitM phase is more than one, we perform an exhaustive search on additional plaintext-ciphertext pairs to get the correct key.

The standard MitM attack decomposes a cipher into two subciphers with two independent keys k_f and k_b. The attack was generalized to multiple subciphers with more than two matching phases by Zhu and Gong [141]. An illustration of multi-dimensional MitM with 4 subciphers is shown in Figure 2.26.

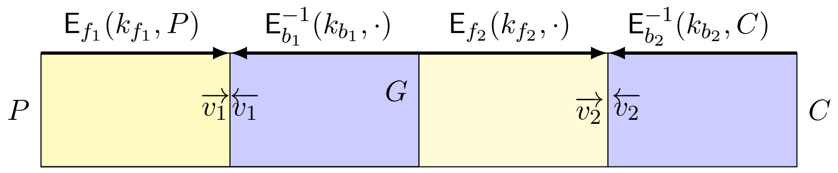

$\mathsf{E}_{f_1}(k_{f_1}, P)$
$\mathsf{E}_{b_1}^{-1}(k_{b_1}, \cdot)$
$\mathsf{E}_{f_2}(k_{f_2}, \cdot)$
$\mathsf{E}_{b_2}^{-1}(k_{b_2}, C)$
P
$\overrightarrow{v_1}$
$\overleftarrow{v_1}$
G
$\overrightarrow{v_2}$
$\overleftarrow{v_2}$
C

Figure 2.26: Multi-dimensional MitM attack with 4 subciphers and one guess value G

2.5 Lightweight Cryptography and Design Principles

Lightweight Cryptography is the study of cryptographic algorithms which are suitable for resource constrained environments such as RFIDs, Smart cards, Internet of Things (IoT) or sensor networks. In particular, it is more about finding a trade-off among the three metrics *security and functionalities*, *resources* and *performance* (Figure 2.27). In the following, we first describe these metrics and then discuss the design principles that we follow for the next part of thesis.

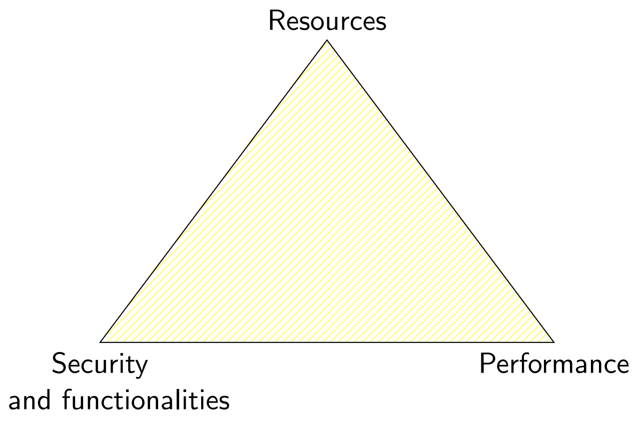


Figure 2.27: Lightweight cryptography metrics

2.5.1 Metrics of lightweight cryptography

Security and functionalities. For a fixed security, are we aiming for a single functionality or multiple functionalities? It could be confidentiality, data integrity and authenticity, hash or psuedorandom number generator.

Resources. *Area* which is the implementation size or physical area (in Gate Equivalents). In case of software, it is *code size* which is the amount of data needed to evaluate the function independent of input (in bytes).

Performance. The performance can be evaluated in terms of *Throughput*, *Latency*, *Power* or *RAM*.

1. *Throughput.* Amount of data processed per time unit (in bits/bytes per second)
2. *Latency.* Time taken to obtain the output of the circuit once its input has been set (in seconds)
3. *Power.* Amount of power needed to use the circuit (in Watts)
4. *RAM.* Amount of data written to memory during each evaluation of the function (in bytes)

The overall goal of lightweight cryptography is to find a solution in Figure 2.27 which can optimize a single metric or a combination of metrics. Obviously, security and resources are directly proportional but not resources and performance. For example, AES software implementation has very low code size but if we want to implement it in hardware within 2000 GE, then its performance degrades. There exist many primitives (Figure 2.28) which try to find a balance of trade-off among the above three metrics.

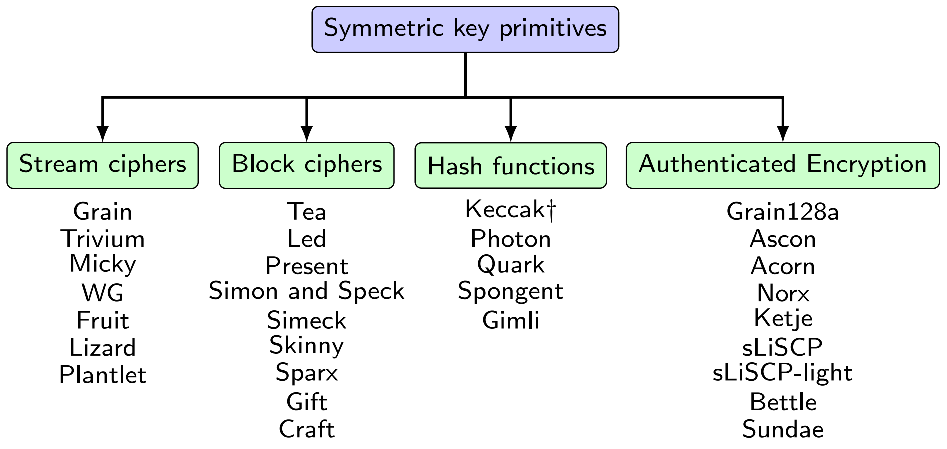


Figure 2.28: Existing lightweight symmetric key primitives. † Keccak with state size 200 bits.

2.5.2 Design principles

Our primary goal is to design secure and lightweight symmetric key primitives. The secondary goal is to achieve multiple functionalities with a low implementation overhead. For instance, to attain AEAD functionality and hash using a block cipher, we have to implement both AEAD mode (GCM/OCB/OCB3 Section 2.3.5) and hash mode (Davies Meyer or Merkle Damgård Section 2.3.3). Moreover, using a block cipher as primitive implies we need registers for both state and key scheduling. For lightweight scenario, this may require more resources. However, sponge based modes can be easily adapted to meet these requirements and do not require key scheduling.

Since the underlying primitive of sponges is a cryptographic permutation, we aim to design permutations which have low hardware cost. At the same time, they are efficient in software. Most importantly, from a security perspective the permutation should be indistinguishable from a random permutation. In this context, we first look into round functions which have the following properties.

- High algebraic degree
- Low differential probability and square correlation
- High diffusion rate, i.e., a state bit depends on most of the other state bits
- No rotational or symmetric properties. If there is one, we look for an efficient method to break it.

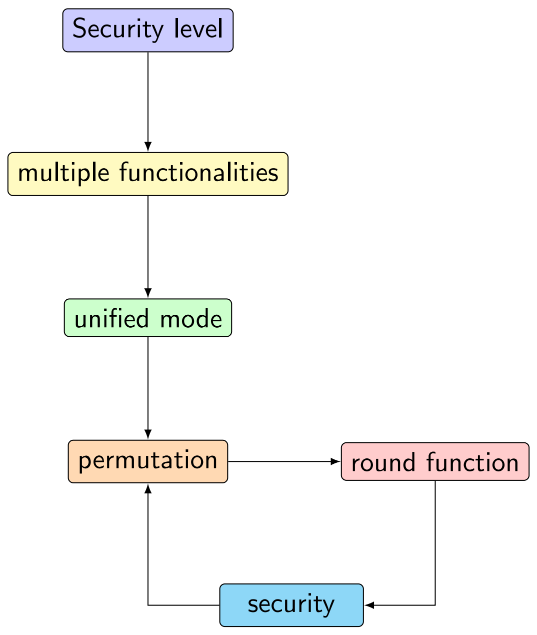


Figure 2.29: Flowchart of design approach

Next, we choose the number of rounds of permutation by analyzing its differential, linear and algebraic properties using MILP and SAT/SMT solvers. More precisely, as a designer we ensure that there exists no distinguisher with probability better than $2^{-b/2}$ where b is the permutation size in bits. Our entire design approach is illustrated in Figure 2.29.

In what follows, we present the specifications of lightweight ciphers whose underlying structures with varying tweaks are used in the design of permutations proposed in Chapters 3-5.

2.5.3 Simon-like block biphers

Simon-$2n/mn$ where $2n$ and mn denote the blocksize and key length, respectively, is a family of block ciphers proposed by NSA in 2013 [23]. A generic diagram of a Simon-like block cipher is depicted in Figure 2.30. It adopts an NLFSR [67] based structure where the nonlinearity comes from the quadratic function $f_{(a,b,c)}(x) = \mathsf{L}^a(x) \& \mathsf{L}^b(x) + \mathsf{L}^c(x)$. We refer to $f_{(a,b,c)}$ as a Simon-like nonlinear function unless the shift parameter set (a, b, c) is explicitly mentioned.

For an r-round cipher, the $(i + 2)$-th element of NLFSR sequence is given by

$$s_{i+2} = f_{(a,b,c)}(s_{i+1}) + s_i + k_i$$

where $k_i \in \mathbb{F}_2^n$ is the i-th round subkey[7] and $0 \le i < r$. Finally, the ciphertext is the r-th state of NLFSR, i.e., (s_{r+1}, s_r). The shift parameters of Simon are given by $(a, b, c) = (8, 1, 2)$.

Simeck-$2n/mn$ was proposed in CHES 2015 by Yang *et al.* [135] and adopts a Simon-

[7] $k_0, k_1, \ldots, k_{m-1}$ are first m n-bit words of key

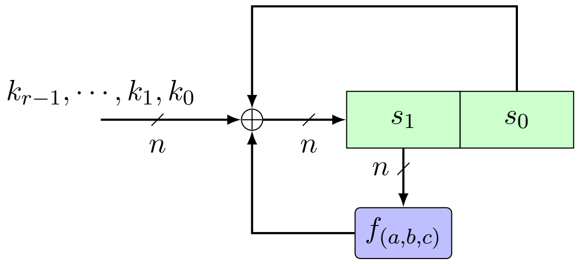


Figure 2.30: Simon-like block cipher

like structure with the shift parameters given by $(5, 0, 1)$. In a way, it has more efficient and compact hardware implementation because of reuse of the round function in the key scheduling algorithm.

Key scheduling algorithms. For $n = 16$ and $m = 4$, $r = 32$ and the subkeys are calculated as follows.

$$\begin{aligned} \text{Simon-32/64}: \quad & k_{i+4} = Z_i + k_i + k_{i+1} + \mathsf{L}^{15}(k_{i+1}) + \mathsf{L}^{13}(k_{i+3}) + \mathsf{L}^{12}(k_{i+3}), \\ \text{Simeck-32/64}: \quad & k_{i+4} = Z_i + f_{(5,0,1)}(k_{i+1}) + k_i, \end{aligned}$$

where Z_i denotes the i-th round constant.

2.5.4 Welch-Gong stream ciphers

The Welch-Gong (WG) stream cipher family is a hardware oriented stream cipher based on word-oriented NLFSR over extension fields [101]. We denote an instance of WG stream cipher by WG-m where m is the dimension of the finite field over $\mathbb{F}_2$, i.e., base field. Figure 2.31 shows a generic structure of WG-m and the individual components are described below.

1. **Base field defining polynomial.** Degree m primitive polynomial over $\mathbb{F}_2$.

2. **Extension field defining polynomial.** Degree l primitive polynomial over $\mathbb{F}_{2^m}$.

3. **WG permutation (WGP).** Let $m \not\equiv 0 \bmod 3$, $3k \equiv 1 \bmod m$ and $\gcd(d, 2^m - 1) = 1$. Then the function WGP : $\mathbb{F}_{2^m} \to \mathbb{F}_{2^m}$ is given by WGP$(x) = h(x + 1) + 1$ where $h(x) = x + x^{q_1} + x^{q_2} + x^{q_3} + x^{q_4}$ and q_i's are given by

$$\begin{aligned} q_1 &= 2^k + 1 \\ q_2 &= 2^{2k} + 2^k + 1 \\ q_3 &= 2^{2k} - 2^k + 1 \\ q_4 &= 2^{2k} + 2^k - 1. \end{aligned}$$

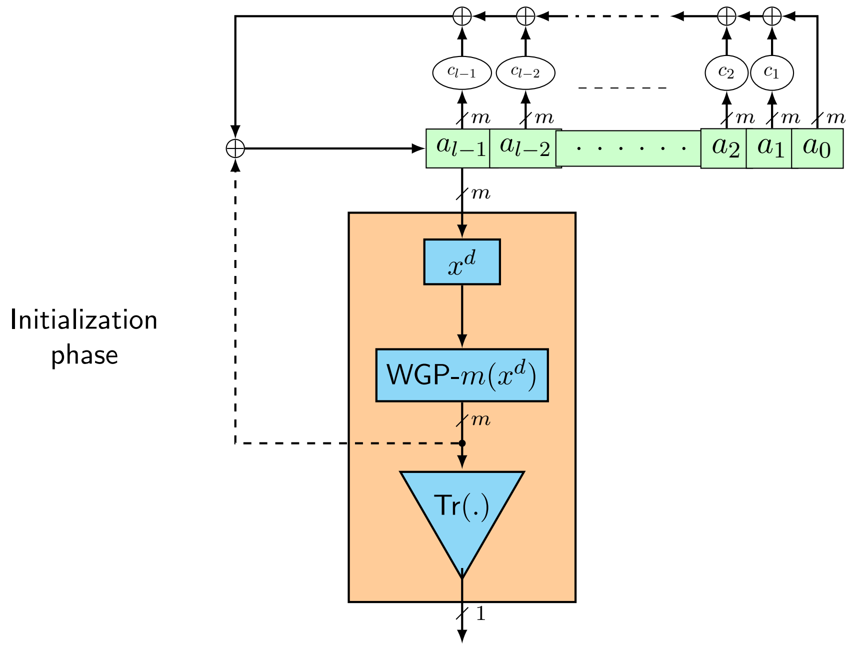


Figure 2.31: Generic structure of WG stream cipher

More about the selection of optimal parameters for WGP is discussed in [94].

4. **Trace function.** The Trace function $\mathsf{Tr} : \mathbb{F}_{2^m} \to \mathbb{F}_2$ is given by

$$\mathsf{Tr}(x) = x + x^2 + \cdots + x^{2^{m-1}}.$$

The cipher runs in two phases: *key initialization phase* and *key generation phase*. During the key initialization phase, the state is first loaded with key and nonce, and then updated for $2l$ clock cycles with nonlinear feedback (output of WGP$(\cdot)$ is feedback for updating the state). No output is generated for first $2l$ clock cycles. After that, the key generation phase starts where a single bit is taken as output and the state is updated linearly. This is repeated until desired number of keystream bits is obtained.

Cryptographic properties of WG stream cipher. The keystream sequence of WG-m has the following properties.

1. Period is $2^{ml} - 1$.
2. It is balanced.
3. It has an ideal 2-level autocorrelation property.
4. Any t-tuple is equally likely distributed (ideal t-tuple distribution) for $1 \leq t \leq l$.

5. Linear complexity of the keystream increases exponentially with m.

Family members. The first family member WG-29 [100] proceeded to Phase 2 of the eSTREAM competition. Later, the lightweight variants WG-5 [9], WG-7 [93] and WG-8 [63] were proposed for constrained environments, and WG-16 [142, 64, 62] was proposed for 4G LTE.

Part I

Design of Lightweight Cryptographic Permutations

Chapter 3

sLiSCP and sLiSCP-light

Contents

Declaration of Contributions

This chapter is based on [14, 16, 15]. My main contributions are as follows.

- Equal contribution in design of both permutations.
- Analysis of the differential and linear properties of Simeck sbox using the SAT/SMT tool [86]. MILP model for bounding the minimum number of active Simeck sboxes.
- Modeled the word-based division property of sLiSCP and the bit-based division property of sLiSCP-light using MILP to find algebraic degree bounds, integral and zero-sum distinguishers.

3.1 Introduction

Ever since the introduction of sponge functions in 2008 by Bertoni *et al.* [31], there has been a surge in the design of sponge based cryptographic primitives. The main reason being a cryptographic permutation utilizing sponge construction can be easily transformed to AEAD, Hash, MAC or Pseduorandom Bit Generator (PRBG) (Sections 2.3.3-2.3.5) [36, 35, 33]. As a result, there is a natural inclination towards designing a cryptographic permutation rather than designing a specific mode. Consequently, starting from the Keccak family of permutations [37], several lightweight cryptographic permutations such as Quark [17], Photon [70], Spongent [41], Norx [19], Ascon [60] and Gimli [29] have been proposed. Our goal here is to design permutations that can achieve more efficient hardware and software performances than the existing permutations at the same security level.

In this chapter, we present the design and security analysis of two lightweight cryptographic permutations, namely sLiSCP and sLiSCP-light. The design of these permutations utilizes two simple and hardware efficient components, namely 4-branch Type-II GFS (Section 2.3.2.2) and unkeyed reduced-round Simeck block cipher (Section 2.5.3). These two structures are well studied in the literature and makes our security analysis easier. We analyze the security of sLiSCP and sLiSCP-light with respect to distinguishing attacks such as differential and linear, and integral distinguishers.

Outline. The rest of the chapter is organized as follows. Section 3.2 provides the detailed specifications of sLiSCP and sLiSCP-light permutations along with their underlying nonlinear component Simeck sbox. In Section 3.3, we present the detailed security analysis of the Simeck sbox. Finally, in Section 3.4, we extend the analysis of the Simeck sbox to provide security bounds of sLiSCP and sLiSCP-light permutations.

3.2 Specifications of sLiSCP and sLiSCP-light

sLiSCP, Simeck-based Permutations for Lightweight Sponge Cryptographic Primitives is a family of cryptographic permutations which have low hardware implementation cost and are efficient in software. sLiSCP-light is a tweaked variant of sLiSCP. In this section, we present their specifications.

3.2.1 The nonlinear function SB-$[2n, u]$

We use the unkeyed reduced-round Simeck block cipher with block size $2n$ ($n \in \{16, 24, 32\}$)and u rounds as the nonlinear operation of both permutations, and denote it by SB-$[2n, u]$. Below we provide the details of SB-$[2n, u]$, henceforth referred to as *Simeck sbox.*

Definition 3.1 (SB-$[2n, u]$: Simeck sbox)**.** Let $rc = (q_{u-1}, \ldots, q_0)$ where $q_j \in \{0, 1\}$ and $0 \leq j < u$. A Simeck sbox is a permutation of a $2n$-bit input, constructed by iterating the Simeck-$2n$ block cipher for u rounds with round constant addition $\gamma_j = 1^{n-1}||q_j$ in place of key addition.

An illustrated description of the Simeck sbox (as an NLFSR) is shown in Figure 3.1 and is given by

$$(x_{u+1}||x_u) \leftarrow \texttt{SB-}[2n, u](x_1||x_0, rc)$$

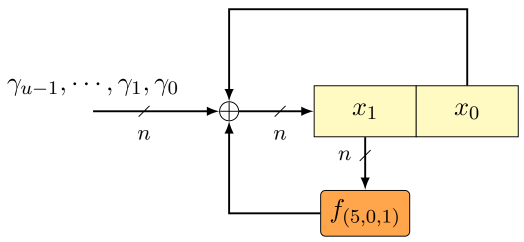


Figure 3.1: Simeck sbox ($\mathtt{SB}$-$[2n, u]$)

where

$$x_j \leftarrow f_{(5,0,1)}(x_{j-1}) \oplus x_{j-2} \oplus \gamma_{j-2},\ \ 2 \le j < u$$

and $f_{(5,0,1)} : \{0,1\}^n \to \{0,1\}^n$ is defined as

$$f_{(5,0,1)}(x) = (\mathsf{L}^5(x) \odot x) \oplus \mathsf{L}^1(x).$$

For the properties of general $f_{(a,b,c)}(x) = \mathsf{L}^a(x) \& \mathsf{L}^b(x) + \mathsf{L}^c(x)$, please see Section 9.3.

3.2.2 Description of sLiSCP

sLiSCP is an iterative permutation that takes a b-bit input and produces an output of b bits where $b = 4 \times 2n$ and $n \in \{24, 32\}$. We denote by sLiSCP-b a b-bit sLiSCP permutation. The state is divided into four $2n$-bit blocks X_0^i, X_1^i, X_2^i and X_3^i where i denotes the state at the beginning of the i-th step. As shown in Figure 3.2, the i-th step of sLiSCP-b is evaluated in three sub-steps which are described below.

1. Store the original values of X_1^i and X_3^i in temporary registers Y_1^i and Y_3^i, and then update X_1^i and X_3^i by applying $\mathtt{SB}$-$[2n, u]$ with round constants rc_0^i and rc_1^i, respectively.

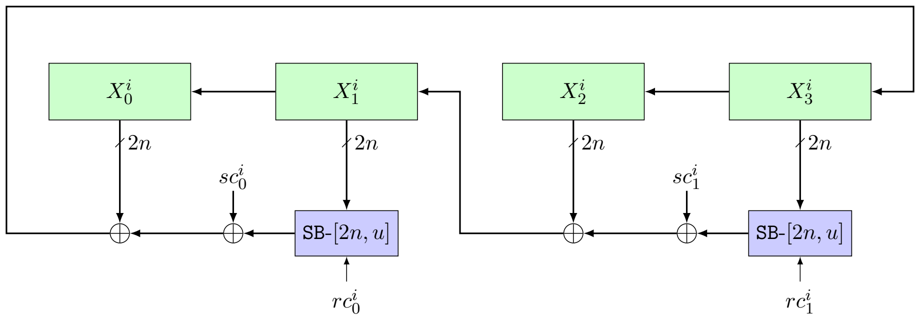
$$\begin{aligned} Y_1^i &\leftarrow X_1^i \\ Y_3^i &\leftarrow X_3^i \\ X_1^i &\leftarrow \mathtt{SB}\text{-}[2n, u](X_1^i) \\ X_3^i &\leftarrow \mathtt{SB}\text{-}[2n, u](X_3^i) \end{aligned}$$

2. Mix the blocks and add the step constant tuple (sc_0^i, sc_1^i), i.e.,

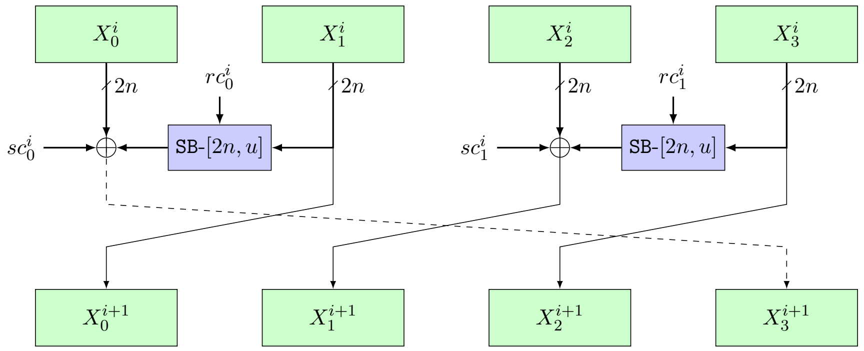
$$\begin{aligned} X_0^i &\leftarrow X_0^i \oplus sc_0^i \oplus X_1^i \\ X_2^i &\leftarrow X_2^i \oplus sc_1^i \oplus X_3^i \end{aligned}$$

3. Blockwise left cyclic shuffle, i.e., $(0, 1, 2, 3) \to (1, 2, 3, 0)$. More precisely,

$$\begin{aligned} X_0^{i+1} &\leftarrow Y_1^i \\ X_1^{i+1} &\leftarrow X_2^i \\ X_2^{i+1} &\leftarrow Y_3^i \\ X_3^{i+1} &\leftarrow X_0^i \end{aligned}$$



(a) NLFSR structure



(b) Type-II GFS

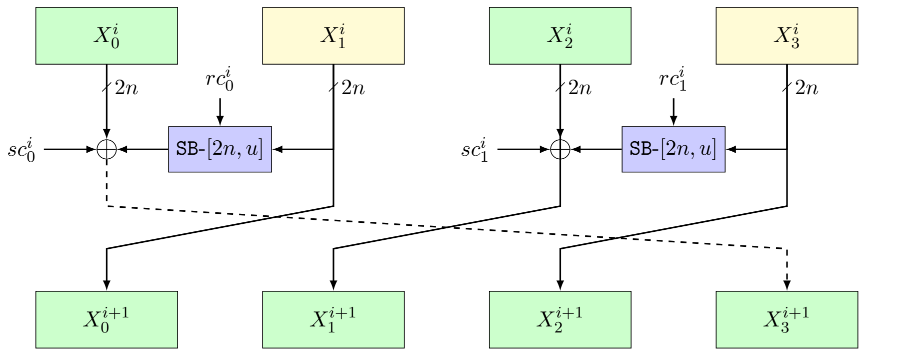
Figure 3.2: Step function of sLiSCP where (a) NLFSR structure and (b) Type-II GFS

3.2.3 Towards sLiSCP-light

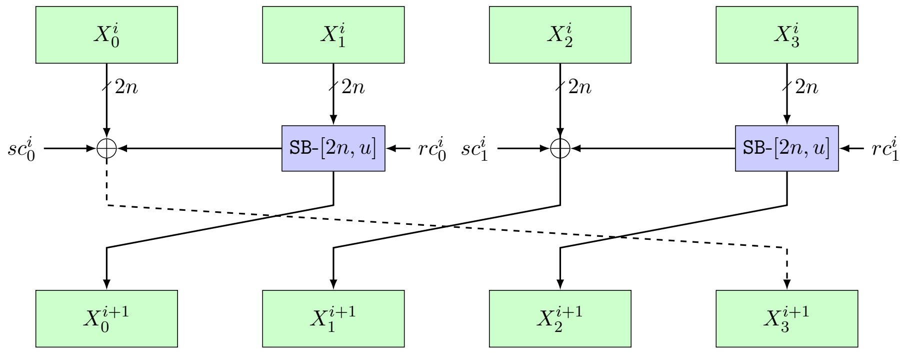
A careful look at the design of sLiSCP's step function reveals that two extra temporary registers, each of size $2n$ bits (shown in yellow color in Figure 3.3 (a)) are needed to store the values of odd indexed blocks at each step. This is due to Type-II GFS round function which is invertible by design, and does not require the invertibility of the nonlinear component. In our case, the nonlinear function SB-$[2n, u]$ is a permutation. Thus, for lightweight applications, this extra hardware overhead of $2 \times 2n$ bits register for temporary storage is unjustified. As a result, we look for a solution to remove this overhead without lowering the overall security.

We observe that a simple tweak, i.e., changing the position of Simeck sboxes (Figure 3.3 (b)) could solve the above problem. However, we are unware of any similar construction except Skipjack Rule A [40]. Thus, a thorough security analysis of this new design is required which we discuss in Section 3.4.

Description of sLiSCP-light. sLiSCP-light is a tweaked variant of sLiSCP. It utilizes the same components as of sLiSCP, i.e., Simeck sboxes, round and step constants. The only difference is in the step function (Figure 3.3 (b)) where the positions of Simeck sboxes are changed. The step function is computed in three steps described below.



(a) sLiSCP permutation



(b) sLiSCP-light permutation

Figure 3.3: Step functions of sLiSCP and sLiSCP-light

1. Application of SB-$[2n, u]$ to odd indexed blocks X_1^i and X_3^i with round constants rc_0^i and rc_1^i, respectively.

$$X_1^i \leftarrow \texttt{SB-}[2n, u](X_1^i)$$
$$X_3^i \leftarrow \texttt{SB-}[2n, u](X_3^i)$$

2. Mix the blocks and add the step constants (sc_0^i, sc_1^i), i.e.,

$$X_0^i \leftarrow X_0^i \oplus sc_0^i \oplus X_1^i$$
$$X_2^i \leftarrow X_2^i \oplus sc_1^i \oplus X_3^i$$

3. Blockwise left cyclic shuffle, i.e., $(0, 1, 2, 3) \rightarrow (1, 2, 3, 0)$. More precisely,

$$X_0^{i+1} \leftarrow X_1^i$$
$$X_1^{i+1} \leftarrow X_2^i$$
$$X_2^{i+1} \leftarrow X_3^i$$
$$X_3^{i+1} \leftarrow X_0^i$$

3.2.4 Permutation instances

Table 3.1 presents the recommended parameters for two lightweight instances of the sLiSCP and sLiSCP-light permutations.

Table 3.1: Recommended parameter sets for sLiSCP and sLiSCP-light permutations

Permutation (b-bit)	$2n$	Rounds u	Steps s	Total # rounds ($u \times s$)
sLiSCP-192	48	6	18	108
sLiSCP-256	64	8	18	144
sLiSCP-light-192	48	6	12	72
sLiSCP-light-256	64	8	12	96

Remark 3.1. The number of steps for sLiSCP-light is 12 compared to 18 in sLiSCP. Moreover, we do not require additional two $2n$ bit registers. Thus, sLiSCP-light reduces the hardware area of sLiSCP by 16% and at the same time improves the throughput by 30%.

3.2.5 Round and step constants

In Tables 3.2 and 3.3, we list the hex values of the constants. Each step constant is appended with $1^{40}\|00$ and 1^{56} for $b = 192$ and $b = 256$, respectively. This procedure results in a number of inversions, which break the propagation of the rotational property in one step.

Table 3.2: Round and step constants when $b = 192$

Step i	(rc_0^i, rc_1^i)	(sc_0^i, sc_1^i)
0 - 5	(7, 27), (4, 34), (6, 2e), (25, 19), (17, 35), (1c, f)	(8, 29), (c, 1d), (a, 33), (2f, 2a), (38, 1f), (24, 10)
6 - 11	(12, 8), (3b, c), (26, a), (15, 2f), (3f, 38), (20, 24)	(36, 18), (d, 14), (2b, 1e), (3e, 31), (1, 9), (21, 2d)
12 - 17	(30, 36), (28, d), (3c, 2b), (22, 3e), (13, 1), (1a, 21)	(11, 1b), (39, 16), (5, 3d), (27, 3), (34, 2), (2e, 23)

Table 3.3: Round and step constants when $b = 256$

Step i	(rc_0^i, rc_1^i)	(sc_0^i, sc_1^i)
0 - 5	(f, 47), (4, b2), (43, b5), (f1, 37), (44, 96), (73, ee)	(8, 64), (86, 6b), (e2, 6f), (89, 2c), (e6, dd), (ca, 99)
6 - 11	(e5, 4c), (b, f5), (47, 7), (b2, 82), (b5, a1), (37, 78)	(17, ea), (8e, 0f), (64, 04), (6b, 43), (6f, f1), (2c, 44)
12 - 17	(96, a2), (ee, b9), (4c, f2), (f5, 85), (7, 23), (82, d9)	(dd, 73), (99, e5), (ea, 0b), (0f, 47), (04, b2), (43, b5)

3.3 Security Analysis of Simeck Sbox

In this section, we discuss the security of SB-$[2n, u]$ which is the nonlinear function of sLISCP and sLISCP-light permutations.

3.3.1 Differential and linear properties

Our goal is to find a u-round differential $(\alpha_1 \| \alpha_0, \alpha_{u+1} \| \alpha_u)$ $(\alpha_i \in \mathbb{F}_2^n)$ with maximum probability. Since the block size is $2n$, an exhaustive search requires 2^{6n} u-round evaluations of SB-$[2n, u]$. For example, the case of $n = 24$ and $u = 6$ implies 2^{144} 6-round evaluations of SB-$[48, 6]$, which is infeasible. We present two different approaches to estimate an upper bound of u-round differential probability (Definition 2.3). We denote it by Maximum Estimated Differential Probability (MEDP).

3.3.1.1 Approach 1

We assume that SB-$[2n, u]$ is a Markov cipher (Definition 2.6). The procedure to compute MEDP is given in Algorithm 3.1.

Algorithm 3.1 Approach 1 to compute MEDP

```
1: Extract all $u$-round optimal differential characteristics (Definition 2.7) using SAT/SMT
   tool [86]. Denote this set by $\Delta_u$.
2: Take an empty list LIST. For each $(\alpha_0, \alpha_1, \ldots, \alpha_{u+1}) \in \Delta_u$, take the differential pair
   $(\alpha_1 \| \alpha_0, \alpha_{u+1} \| \alpha_u)$, compute the $r$-round differential probability and append it to the LIST.
3: Assign MEDP(SB-$[2n, u]$) as the maximum value of LIST.
```

The obtained results using Algorithm 3.1 are given in Tables 3.4 and 3.5.

Table 3.4: Optimal differential characteristic probability and linear characteristic square correlation for $\texttt{SB}\text{-}[2n,u]$ where $2n = 32$, 48 and 64. The values are given in the $\log_2(\cdot)$ scale.

Rounds (u)	1	2	3	4	5	6	7	8	9	10	11	12	13
$\texttt{SB}\text{-}[2n,u]$	0	-2	-4	-6	-8	-12	-14	-18	-20	-24	-26	-30	-32

Table 3.5: MEDP for $\texttt{SB}\text{-}[2n,u]$ for $2n = 32$, 48 and 64

Rounds (u)	1	2	3	4	5	6	7	8	9
MEDP	0	-2	-4	-6	-8	-11.3	-13.3	-16.6	-18.6

3.3.1.2 Approach 2

In the previous approach, we have adopted the conventional Markov assumption. However, this is not true as Simeck sboxes are parameterized by a set of fixed round constants. Thus, we use an alternative approach to derive tight upper bounds on the MEDP of the constant-based Simeck sboxes. Our approach is based on the following two important observations on the differential properties of different block sizes of Simeck round function.

1. The probabilities of optimal differential characteristics are exactly equal for reduced-round Simeck with block sizes 32, 48, and 64 bits (Table 3.4).

2. There exist optimal differential characteristics in Simeck-32 which are related to optimal differential characteristics in Simeck-48 and Simeck-64. One such example is given in Table 3.6 where we only add zeros at specific positions to match the block size.

We first compute the MEDP of $\texttt{SB}\text{-}[32,6]$ and $\texttt{SB}\text{-}[32,8]$ following Algorithm 3.2, and then use these results to compute the MEDP of $\texttt{SB}\text{-}[48,6]$. For $\texttt{SB}\text{-}[64,8]$, we conjecture its MEDP value.

We find that $\Delta_6 = 3072$ and $\Delta_8 = 2560$. Tables 3.7 and 3.8 depict the sets of maximum probability differentials parametrized by the round constants (given in hex) for $\texttt{SB}\text{-}[32,6]$ and $\texttt{SB}\text{-}[32,8]$, respectively. Thus, we set $\text{MEDP}(\texttt{SB}\text{-}[32,6]) = 2^{-10.35}$ and $\text{MEDP}(\texttt{SB}\text{-}[32,8]) = 2^{-15.4}$.

MEDP of $\texttt{SB}\text{-}[48,6]$ and $\texttt{SB}\text{-}[48,8]$. In Tables 3.7 and 3.8, we observe that there are six (resp. eleven) sets for which each set has the same differential probability. Given that the optimal differential characteristic probabilities of u-round Simeck sbox are equal (Table 3.4) when the block sizes are 32, 48, and 64 bits, we pick one differential candidate, i.e., differentials

Table 3.6: Related optimal differential characteristics of Simeck with block sizes 32, 48 and 64

	Simeck-32		Simeck-48		Simeck-64		
Round (u)	α_{u+1}	α_u	α_{u+1}	α_u	α_{u+1}	α_u	$\log_2(p)$
0	0x0001	0x0002	0x00**00**01	0x00**00**02	0x00**0000**01	0x00**0000**02	-2
1	0x0000	0x0001	0x00**00**00	0x00**00**01	0x00**0000**00	0x00**0000**01	-0
2	0x0001	0x0000	0x00**00**01	0x00**00**00	0x00**0000**01	0x00**0000**00	-2
3	0x0002	0x0001	0x00**00**02	0x00**00**01	0x00**0000**02	0x00**0000**01	-2
4	0x0005	0x0002	0x00**00**05	0x00**00**02	0x00**0000**05	0x00**0000**02	-4
5	0x0008	0x0005	0x00**00**08	0x00**00**05	0x00**0000**08	0x00**0000**05	-2
6	0x0015	0x0008	0x00**00**15	0x00**00**08	0x00**0000**15	0x00**0000**08	-

Algorithm 3.2 Approach 2 to compute MEDP

```
1: For SB-$[32,6]$, extract all 6 rounds optimal differential characteristics. Denote this set by $\Delta_6$.
2: For each round constant and for each $(\alpha_0, \alpha_1, \ldots, \alpha_7) \in \Delta_6$, we calculate the exact differential probability of 6-round differential $(\alpha_1 \| \alpha_0, \alpha_7 \| \alpha_6)$ by running a parallel exhaustive search for determining the number of solutions (out of the $2^{32}$ possible inputs). We then set the MEDP for each Simeck sbox to the maximum probability among all the tested differentials.
3: Repeat the same procedure for SB-$[32,8]$.
```

marked with blue color in Tables 3.7 and 3.8. Next, we check if it is one of the differentials associated to the extracted optimal characteristic. Then, for these selected differentials, we run a parallel exhaustive search to get their exact differential probability. Table 3.9 shows the probabilities of selected differentials.

MEDP of SB-$[64,8]$. We note that the differential probabilities of Simeck sboxes with 48-bit block size are slightly lower than that of 32-bit block size. Since it is computationally infeasible for us to run an exhaustive search on 64-bit blocks, we conjecture that the MEDP of SB-$[64,8]$ sbox is also slightly lower than that of SB-$[48,8]$. Accordingly, we set MEDP(SB-$[64,8]$) to be equal to the one we extracted for SB-$[48,8]$ which is equal to $2^{-15.86}$.

Details of the experimental setup. All the experiments are conducted on a server with the following specifications: 8 cores per node, 16 GB RAM per node, Intel Xeon E5540 @2.53 GHz, 64 bit Linux Centos 6.4 OS.

Table 3.7: Differentials with maximum probability for SB-$[32, 6]$ sboxes. Here time denotes the average time to find all solutions of one differential.

Round constants	$(\alpha_1 \| \alpha_0, \alpha_7 \| \alpha_6)$	# solutions	DP $(\log_2(.))$	Time(s)
7, 27, 4, 6, 25, 26, 24	(04000a00, 1a000a00), (04000a00, 5a000a00), (04000a00, 52000a00), (04000a00, 12000a00)	3293184	-10.348948	324.75
34, 17, 35, 15	(04000a00, 1a010a00), (04000a00, 5a010a00), (04000a00, 52010a00), (04000a00, 12010a00)	3293184	-10.348948	323.62
19, 8, 38	(0a001a00, 0a000400), (0a005a00, 0a000400), (0a005200, 0a000400), (0a001200, 0a000400)	3293184	-10.348948	324.26
3b, a	(0a001a01, 0a000400), (0a005a01, 0a000400), (0a005201, 0a000400), (0a001201, 0a000400)	3293184	-10.348948	323.92
12, 20	(40014023, 40018000), (4001402b, 40018000), (80004001, 402b4001), (80004001, 40234001)	3280896	-10.354342	324.03
2e, 1c, 1f, c, 2f, 3f	(00020005, 00890005), (00018002, 80548002), (80004001, 402a4001), (20005000, 900a5000), (00018002, 80448002), (00020005, 00a90005), (0004000a, 0152000a), (4000a000, 2011a000), (80004001, 40224001), (0004000a, 0112000a), (20005000, 90085000), (00800140, 22400140), (10002800, 48052800), (10002800, 48042800), (05008900, 05000200), (000500a9, 00050002), (00200050, 08900050), (4001402a, 40018000), (4000a000, 2015a000), (0a001201, 0a000400), (00800140, 2a400140), (00080014, 02a40014), (00080014, 02240014), (5000900a, 50002000), (08001400, 24021400), (08001400, a4021400), (28004805, 28001000), (80028054, 80020001), (0500a900, 05000200), (00280548, 00280010), (80028044, 80020001), (50009008, 50002000), (28004804, 28001000), (40014022, 40018000), (a0002011, a0004000), (00050089, 00050002), (02804480, 02800100), (004000a0, 150000a0), (a0002015, a0004000), (000a0112, 000a0004), (004000a0, 110000a0), (00100028, 04680028), (04000a00, 12000a00), (02805480, 02800100), (000a0152, 000a0004), (00100028, 05680028), (1400a402, 14010800), (00200050, 0a900050), (14002402, 14010800), (0a005201, 0a000400), (04000a00, 52000a00), (01402240, 01400080), (001402a4, 00140008), (00140224, 00140008), (01000280, 54800280), (01000280, 44800280), (00a01520, 00a00040), (01402a40, 01400080), (00a01120, 00a00040), (00500890, 00700020), (00280448, 00280010), (02000500, a9000500), (00500a90, 00700020), (02000500, 89000500)	3231744	-10.376119	281.97

Table 3.8: Differentials with maximum probability for SB-$[32, 8]$ sboxes

Round constants	$(\alpha_1\|\alpha_0, \alpha_9\|\alpha_8)$	# solutions	DP ($\log_2(.)$)	Time(s)
b	(0a001201, 3a001000)	99232	-15.401482	405.02
f, 47, 4, 96, 7	(00a011a0, 02a00100)	90418	-15.535678	404.84
b2, a1	(010002a0, 118000a0)	90418	-15.535678	404.88
f1, 73	(010002a0, 11a000a0)	90418	-15.535678	404.86
78	(10002a00, 5a010a00)	89842	-15.544898	406.29
44, 4c	(00a01180, 02a00100)	87974	-15.575210	396.84
43	(00a011a0, 02a00100), (010002a0, 11a000a0)	87264	-15.586901	405.95
82	(00a011a0, 02a00100), (010002a0, 118000a0)	87264	-15.586901	398.71
e5	(80005001, d0085000), (5000d008, 50018000)	81282	-15.689352	405.10
b5, 37, f5,	(80005001, d0085000)	79488	-15.721551	405.02
ee	(5000d008, 50018000)	79488	-15.721551	404.39

Table 3.9: Differential probabilities of selected differentials for SB-$[48, u]$

Rounds (u)	Round constants	Selected differential	DP ($\log_2(.)$)
6	27	(0400000a0000, 1a00000a0000)	-10.66
	15	(000400000a00, 011a00000a00)	-10.66
	8	(0a00001a0000, 0a0000040000)	-10.66
	3b	(0a00001a0001, 0a0000040000)	-10.66
	12	(400001400023, 400001800000)	-10.66
	2e	(000200000500, 008900000500)	-10.66
8	b	(0a0000120001, 3a0000100000)	-15.86
	f	(00a00011a000, 02a000010000)	-15.93
	b2	(01000002a000, 01800000a000)	-15.95
	f1	(01000002a000, 11a00000a000)	-15.93
	78	(1000002a0000, 5a00010a0000)	-15.95
	4c	(00a000118000, 02a000010000)	-15.93
	43	(00a00011a000, 02a000010000)	-15.93
	82	(00a00011a000, 02a000010000)	-15.93
	e5	(800000500001, d00008500000)	-15.93
	b5	(800000500001, d00008500000)	-15.91
	ee	(500000d00008, 500001800000	-15.91

3.3.1.3 Notes on linear properties

The analysis of linear properties is exactly same as the differential analysis. The goal here is to find a u-round linear hull $(\alpha_1\|\alpha_0, \alpha_{u+1}\|\alpha_u)$ $(\alpha_i \in \mathbb{F}_2^n)$ with maximum square correlation (Definition 2.8). We first find the optimal linear characteristics square correlation values (Table 3.4). Next, we estimate an upper bound of u-round linear hull square correlation, and denote it by Maximum Estimated Linear Hull Square Correlation (MELHSC). We followed similar approaches (Sections 3.3.1.1 and 3.3.1.2) to compute MELHSC values which are given by

$$\text{MELHSC}(\texttt{SB}\text{-}[48,6]) = 2^{-10.83}$$
$$\text{MELHSC}(\texttt{SB}\text{-}[64,8]) = 2^{-15.64}.$$

3.3.2 Algebraic properties

The round function of Simeck sbox is quadratic (Section 3.2.1) which implies that the algebraic degree of Simeck sbox is at most 2, 4, 8 and 16 after 1, 2, 3 and 4 rounds, respectively. We have computed the algebraic degree using symbolic computation and find that the degrees are in fact 2, 3, 5 and 8. For $u \geq 5$, we are unable to perform symbolic computations because of high density of monomials. Thus, we use a tweaked variant of bit-based division property (Section 2.4.4.1 in Chapter 2) to provide bounds on the algebraic degree when $u \geq 5$. The corresponding MILP model and results are described below.

3.3.2.1 MILP model for bounding algebraic degree

Since the round function consists of only bitwise ANDs and XORs, the linear inequalities can be modeled similarly to the ones mentioned in Section 2.4.4.1. The variables for the i-th round are shown in Figure 3.4 while the entire model for u rounds with tweaks (shown in red color) is explained below.

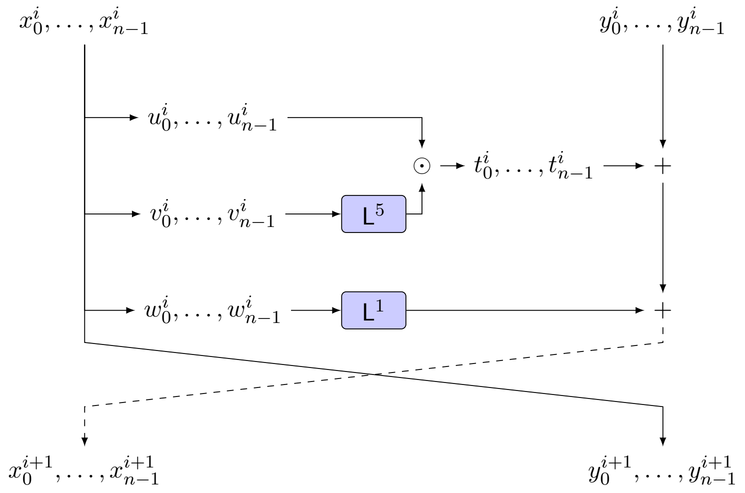

$x_0^i, \ldots, x_{n-1}^i$
$y_0^i, \ldots, y_{n-1}^i$
$u_0^i, \ldots, u_{n-1}^i$
$\odot$
$t_0^i, \ldots, t_{n-1}^i$
$+$
$v_0^i, \ldots, v_{n-1}^i$
L^5
$w_0^i, \ldots, w_{n-1}^i$
L^1
$+$
$x_0^{i+1}, \ldots, x_{n-1}^{i+1}$
$y_0^{i+1}, \ldots, y_{n-1}^{i+1}$

Figure 3.4: i-th round MILP variables for Simeck sbox

MILP model.

1. **Binary variables.** $x_j^i, y_j^i, u_j^i, v_j^i, w_j^i, t_j^i$, $0 \leq j \leq n-1$ and $0 \leq i \leq u-1$

2. **Constraints.** For $0 \leq j \leq n-1$ and $0 \leq i \leq u-1$,

$$\begin{aligned}
x_j^i &= u_j^i + v_j^i + w_j^i + y_j^{i+1} \\
t_j^i &\geq u_j^i \\
t_j^i &\geq v_{j+5 \bmod n}^i \\
x_j^{i+1} &= y_j^i + t_j^i + w_{j+1 \bmod n}^i \\
x_k^u &= 1 \\
x_j^u &= 0 \text{ for } j \in \{0, \ldots, n-1\} \setminus \{k\} \\
y_j^u &= 0
\end{aligned}$$

3. **Objective function.** Maximize $\sum_{j=0}^{n-1} x_j^0 + y_j^0$

In the above model, the first constraint is for the division property of COPY, second and third are for the division property of AND, and fourth is for the division property of XOR. Since we compute the algebraic degree of x_k^u for some k, it is constrained to 1 and the remaining u-th round variables are set to 0. We pass this optimization model to Gurobi solver [2] which returns the maximum value (say d). The solution implies that there exists a degree d monomial term in the ANF representation of x_k^u. Hence, the algebraic degree of x_k^u is d. To compute the algebraic degree of y_k^u, we simply modify the last three constraints (in red color).

3.3.2.2 Results for algebraic degree

The bounds on algebraic degree of Simeck sboxes are given in Table 3.10.

Table 3.10: Algebraic degree bounds of Simeck sboxes

Rounds (u)	1	2	3	4	5	6	7	8
SB-$[48, u]$	2	3	5	8	13	19	27	36
SB-$[64, u]$	2	3	5	8	13	19	27	36

3.3.2.3 MILP model for integral distinguishers

In Table 3.10, if we consider the degree of SB-$[48, 5]$, then it equals 13. This implies a 14-th order derivative of SB-$[48, 5]$ is constant. This means for a $\mathbb{X} \subset \mathbb{F}_2^{48}$ with $|\mathbb{X}| = 2^{14}$, we have

$$\bigoplus_{x \in \mathbb{X}} (\texttt{SB-}[48, 5](x))_j = 0$$

with probability 1 for all j such that $0 \leq j \leq 47$. Here, $\texttt{SB}\text{-}[48,5]_j$ denotes the ANF of j-th component function of $\texttt{SB}\text{-}[48,5]$ after 5 rounds.

In order to find an integral distinguisher using the previous MILP model, we modify it slightly as follows. We consider the simplest example where the 0-th bit of the input state is set as constant (C) and rest as active (A). In particular, we add these additional constraints to the model.

$$\begin{aligned} x_0^0 &= 0 \\ x_j^0 &= 1 \text{ for } 1 \leq j \leq n-1 \\ y_j^0 &= 1 \text{ for } 0 \leq j \leq n-1 \end{aligned}$$

We then evaluate the algebraic degree at the u-th round of j-th component function in terms of the involved active bits. If the algebraic degree equals the number of active bits then the j-th bit is unknown (U), i.e., the value of

$$\bigoplus_{x \in \mathbb{X}} (\texttt{SB}\text{-}[48,u](x))_j$$

is unpredictable. Otherwise, it is balanced (B) in which case

$$\bigoplus_{x \in \mathbb{X}} (\texttt{SB}\text{-}[48,u](x))_j = 0 \tag{3.1}$$

with probability 1. Here $\mathbb{X} = \{0, \star, \star, \cdots, \star\}$ where $\star$ takes values 0 and 1. Accordingly, $|\mathbb{X}| = 2^{47}$.

Remark 3.2. In general, we can choose multiple bits to be constant and modify the model accordingly.

3.3.3 Symmetric properties

The Simeck round function given by $f_{(5,0,1)}(x) = (\mathsf{L}^5(x) \odot x) \oplus \mathsf{L}^1(x)$ is rotationally invariant. More precisely, we have $f_{(5,0,1)}(\mathsf{L}^{\mathsf{i}}(x)) = \mathsf{L}^{\mathsf{i}}(f_{(5,0,1)}(x))$, $\forall\ x \in \mathbb{F}_2^n$. To break this propagation of rotational property, we add round constants rc_0^i and rc_1^i within the Simeck sboxes.

3.4 Security Analysis of sLiSCP and sLiSCP-light

In this section, we analyze the security of the sLiSCP and sLiSCP-light permutations by considering the differential and linear, and algebraic properties.

3.4.1 Differential and linear cryptanalysis

To evaluate upper bounds on differential and linear characteristics of sLiSCP and sLiSCP-light, we follow the Wide Trail Strategy [52] (used by AES designers and subsequently applied to most of the designs). To the best of our knowledge, this is the best bound we can achieve. We first compute the minimum number of differentially and linearly active Simeck sboxes using the MILP model. Next, we use the MEDP and MELHSC bounds of Simeck sboxes (Section 3.3.1) to provide bounds for sLiSCP and sLiSCP-light.

3.4.1.1 MILP model for bounding minimum number of active sboxes

Assumptions. We say a Simeck sbox SB-$[2n, u]$ is active if the input difference to it is non-zero. A non-zero input difference goes to a non-zero output difference with probability 1 as the Simeck sbox is a permutation. For a non-invertible sbox, this holds with some probability but not 1. The XOR operation cancels the difference with probability 2^{-2n}, i.e., for $\alpha, \beta \in \mathbb{F}_2^{2n}$, $\alpha \oplus \beta = 0 \iff \alpha = \beta$. Moreover, a zero difference goes to zero output difference. So, the input difference at the beginning has to be non-zero. The non-zero and zero differences are denoted by integer 1 and 0, respectively.

Since there are 4 blocks in sLiSCP and sLiSCP-light permutations, we have $2^4 - 1$ possible input differences (except the $(0, 0, 0, 0)$ case)). For s steps, there are $2^{2s} \times 15$ paths. The goal is to find a path which has the minimum number of active Simeck sboxes. We model this problem as an optimization problem. Below, we give the exact model for sLiSCP as an example.

MILP model for sLiSCP.

1. **Binary variables.** $x_0, x_1, x_2, x_3,\ t_0^i, t_1^i$ for $0 \le i \le s-1$, and $x_4, \cdots, x_{4+2(s-1)}, x_{3+2s}$

2. **Constraints.**

Non-zero input difference:	$x_0 + x_1 + x_2 + x_3 \ge 1$
Step 0:	$x_0 + x_1 + x_4 - 2t_0^0 \ge 0,\ t_0^0 \ge x_0,\ t_0^0 \ge x_1,\ t_0^0 \ge x_4$
	$x_2 + x_3 + x_5 - 2t_1^0 \ge 0,\ t_1^0 \ge x_2,\ t_1^0 \ge x_3,\ t_1^0 \ge x_5$
Step 1:	$x_1 + x_5 + x_6 - 2t_0^1 \ge 0,\ t_0^1 \ge x_1,\ t_1^0 \ge x_5,\ t_0^1 \ge x_6$
	$x_3 + x_4 + x_7 - 2t_1^1 \ge 0,\ t_1^1 \ge x_3,\ t_1^1 \ge x_4,\ t_1^1 \ge x_7$
Step $i = 2$ to $s-1$:	$x_{2i} + x_{2i+2} + x_{2i+5} - 2t_0^i \ge 0,\ t_0^i \ge x_{2i},\ t_1^i \ge x_{2i+2},\ t_0^i \ge x_{2i+5}$
	$x_{2i+1} + x_{2i+3} + x_{2i+4} - 2t_1^i \ge 0,\ t_1^i \ge x_{2i+1},\ t_1^i \ge x_{2i+3},\ t_1^i \ge x_{2i+4}$

3. **Objective function.** Minimize: $x_1 + x_3 + \cdots + x_{2s} + x_{2s+1}$

In the above model, the conditions of the form $a+b+c-2t \ge 0$, $t \ge a$, $t \ge b$, $t \ge c$ model the difference propagation via XOR operation, i.e., $a \oplus b = c$. The variables in the objective function are inputs to the Simeck sboxes. The model for sLiSCP-light and computing linearly active sboxes is exactly identical[1]. Table 3.11 presents a lower bound on the minimum number of active sboxes for up to 18 steps which we obtained by solving the above model.

Table 3.11: Lower bounds on the number of active Simeck sboxes for sLiSCP and sLiSCP-light

Step (s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
min. # of active Simeck sboxes	0	1	2	3	4	6	6	7	8	9	10	12	12	13	14	15	16	18

3.4.1.2 Estimated bounds of differential and linear characteristics

From Table 3.11, we note that the minimum number of differential/linear active sboxes are 18 and 12 for $s = 18$ and $s = 12$, respectively. Thus, the estimated upper bounds for differential and linear characteristics for sLiSCP and sLiSCP-light permutation are given as follows.

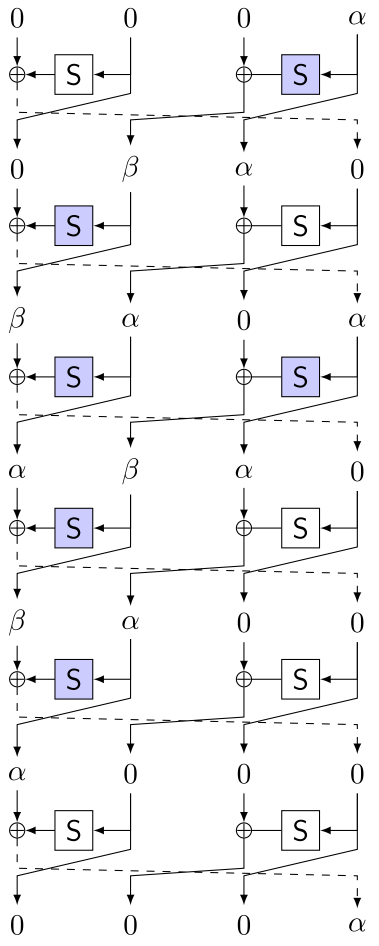
sLiSCP-192:	$(\text{MEDP}(\mathsf{SB}\text{-}[48,6]))^{18}$	$= (2^{-10.66})^{18}$	$= 2^{-191.88}$
sLiSCP-256:	$(\text{MEDP}(\mathsf{SB}\text{-}[64,8]))^{18}$	$= (2^{-15.86})^{18}$	$= 2^{-285.48}$
sLiSCP-light-192:	$(\text{MEDP}(\mathsf{SB}\text{-}[48,6]))^{12}$	$= (2^{-10.66})^{12}$	$= 2^{-127.92}$
sLiSCP-light-256:	$(\text{MEDP}(\mathsf{SB}\text{-}[64,8]))^{12}$	$= (2^{-15.86})^{18}$	$= 2^{-190.32}$
sLiSCP-192:	$(\text{MELHSC}(\mathsf{SB}\text{-}[48,6]))^{18}$	$= (2^{-10.83})^{12}$	$= 2^{-194.94}$
sLiSCP-256:	$(\text{MELHSC}(\mathsf{SB}\text{-}[64,8]))^{18}$	$= (2^{-15.64})^{12}$	$= 2^{-281.52}$
sLiSCP-light-192:	$(\text{MELHSC}(\mathsf{SB}\text{-}[48,6]))^{12}$	$= (2^{-10.83})^{12}$	$= 2^{-129.96}$
sLiSCP-light-256:	$(\text{MELHSC}(\mathsf{SB}\text{-}[64,8]))^{12}$	$= (2^{-15.64})^{12}$	$= 2^{-187.68}$

3.4.1.3 Notes on differential property of sLiSCP and sLiSCP-light

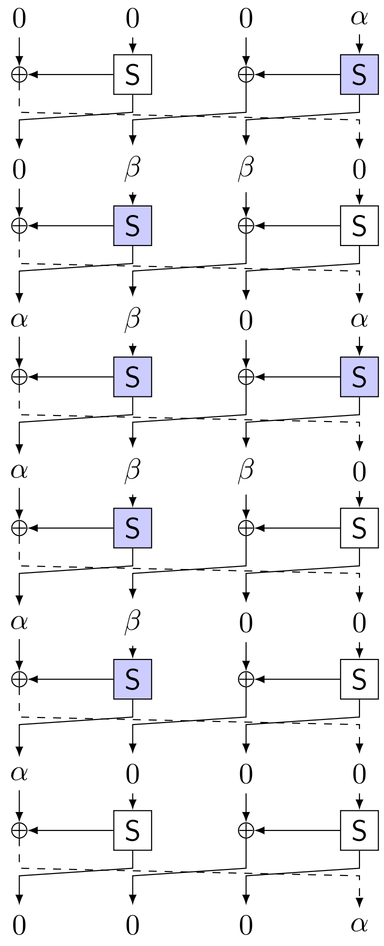
We consider the propagation of a six step differential characteristic to analyze which permutation offers better resistance against differential cryptanalysis. Let the input difference $\alpha \neq 0$ to $\mathsf{SB}\text{-}[2n, u]$ go to output difference β with probability p_1 and β goes to α with probability p_2. As shown in Figure 3.5, the input and output differences to both permutations are $(0, 0, 0, \alpha)$ after 6-steps. In fact, it is a 6-step cycle. Also, note that the number of active sboxes is 6 (active sboxes shown in blue color). However, the differential characteristics probabilites differ which are given as follows.

$$\begin{array}{lll} \text{sLiSCP:} & (\mathsf{Pr}(\alpha,\beta))^4 \times (\mathsf{Pr}(\beta,\alpha))^2 & = p_1^4 p_2^2 \\ \text{sLiSCP-light:} & (\mathsf{Pr}(\alpha,\beta))^2 \times (\mathsf{Pr}(\beta,\alpha))^4 & = p_1^2 p_2^4 \end{array}$$

[1]For example, if XOR is replaced by AES MixColumns, then one has to be careful.



(a) sLiSCP



(b) sLiSCP-light

Figure 3.5: 6-step differential characteristic for (a) sLiSCP and (b) sLiSCP-light where S is SB-$[2n, u]$

As a concrete example for SB-$[48, 6]$, we find $\alpha =$ `0x010000000000` and $\beta =$ `0x1D0000060000` with $p_1 = 2^{-17.8}$ and $p_2 = 2^{-16.3}$. Thus, $p_1^4 p_2^2 = 2^{4\times -17.8 + 2\times 16.3} \approx 2^{-103.8}$ and $p_1^2 p_2^4 = 2^{2\times -17.8 + 4\times 16.3} \approx 2^{-100.8}$. Since for sLiSCP-light-192 we found a differential characteristic whose probability is greater than sLiSCP-192, we argue that sLiSCP-light is slightly weaker than sLiSCP in terms of differential properties. Furthermore, the same observation applies to linear characteristics.

3.4.2 Algebraic properties

To find bounds on the algebraic degree of sLiSCP-light, we extend the MILP model of the Simeck sbox (Section 3.3.2.1) to s steps of the permutation. Table 3.12 provides an upper bound on the algebraic degree of each component function of sLiSCP-light instances.

Table 3.12: Upper bounds on the algebraic degree of sLiSCP-light

		Component function							
sLiSCP-light-192	steps (s)	0-23	24-47	48-71	72-95	96-119	120-143	144-167	168-191
	1	19	13	19	13	19	13	19	13
	2	57	51	57	51	57	51	57	51
	3	129	125	129	125	129	125	129	125
	4	178	177	178	177	178	177	178	177
	5	189	189	189	189	189	189	189	189
sLiSCP-light-256	steps (s)	0-31	32-63	64-95	96-127	128-159	160-191	192-223	224-255
	1	36	27	36	27	36	27	36	27
	2	92	83	92	83	92	83	92	83
	3	183	182	183	182	183	182	183	182
	4	247	247	247	247	247	247	247	247

Integral distinguishers. We model a MILP model similar to Section 3.3.2.3 to search for the integral distinguishers. We set the 0-th bit of the input state as constant (C) and rest as active (A). We then evaluate the algebraic degree at the s-th step of each component function in terms of the involved active bits. We find that for sLiSCP-light-192, after the 8-th step, the component functions 0-59, 70-107, 118-191 have degree less than 191, and hence bits 0-59, 70-107, 118-191 are balanced. As for sLiSCP-light-256, bits 0-63, 192-255 are balanced. Thus, 8-step integral distinguishers exist for both sLiSCP-light-192 and sLiSCP-light-256.

Zero-sum distinguishers. This is a variant of integral distinguisher where all the bits of the state are balanced. We note that the maximum number of steps covered by zero-sum distinguishers in one direction is at most 7. This is because integral distinguisher can cover up to 8 steps. For example $CA^{191} \xrightarrow{7 \text{ steps}} B^{192}$, $CA^{255} \xrightarrow{7 \text{ steps}} B^{256}$. Hence, $7 + 7$ steps zero-sum distinguisher exists for sLiSCP-light-192 (resp. sLiSCP-light-256).

Note that the number of steps in sLiSCP-light is 12, but we have found 14-step zero sum distinguishers. To exploit such distinguishers, one has to start from an intermediate state ($s = 7$) of the permutation. Since the intermediate state is never available to the adversary, we emphasize that such distinguishers are not directly exploitable, and hence do not affect the security.

Remarks on integral distinguisher of sLiSCP. The above analysis can be easily applied to sLiSCP. However, we have only modeled the word-based division property of sLiSCP permutation. The 9-step integral distinguisher for sLiSCP-192 is given in Table 3.13.

This suggests that sLiSCP-light has better algebraic properties than sLiSCP. This is also evident from the fact that sLiSCP-light step function mixes the entire state nonlinearly. In case of sLiSCP, the odd index blocks are updated to next step without any mixing.

Table 3.13: 9-step integral distinguisher for sLiSCP-192

Step (s)	Division property
0	{(48, 46, 48, 48)}
1	{(46, 48, 48, 48)}
2	{(10, 48, 48, 48), (29, 48, 48, 47), (48, 48, 48, 46)}
3	{(1, 48, 48, 13), (10, 48, 48, 12), (29, 48, 48, 11), (48, 48, 48, 10), (1, 48, 47, 32), (10, 48, 47, 31), (29, 48, 47, 30), (48, 48, 47, 29), (48, 48, 46, 48)}
4	{(1, 48, 13, 4), (10, 48, 13, 3), (29, 48, 13, 2), (48, 48, 13, 1), (1, 48, 12, 13), (10, 48, 12, 12), (29, 48, 12, 11), (48, 48, 12, 10), (1, 48, 11, 32), (10, 48, 11, 31), (29, 48, 11, 30), (48, 48, 11, 29), (48, 48, 10, 48), (1, 47, 32, 4), (10, 47, 32, 3), (29, 47, 32, 2), (48, 47, 32, 1), (1, 47, 31, 13), (10, 47, 31, 12), (29, 47, 31, 11), (48, 47, 31, 10), (1, 47, 30, 32), (10, 47, 30, 31), (29, 47, 30, 30), (48, 47, 30, 29), (48, 47, 29, 48), (48, 46, 48, 48)}
5	{(1, 13, 0, 4), (1, 12, 13, 4), (10, 13, 0, 3), (10, 12, 13, 3), (29, 13, 0, 2), (29, 12, 13, 2), (48, 13, 0, 1), (48, 12, 13, 1), (1, 12, 12, 13), (10, 12, 12, 12), (29, 12, 12, 11), (48, 12, 12, 10), (1, 12, 11, 32), (10, 12, 11, 31), (29, 12, 11, 30), (48, 12, 11, 29), (48, 12, 10, 48), (1, 11, 32, 4), (10, 11, 32, 3), (29, 11, 32, 2), (48, 11, 32, 1), (1, 11, 31, 13), (10, 11, 31, 12), (29, 11, 31, 11), (48, 11, 31, 10), (1, 11, 30, 32), (10, 11, 30, 31), (29, 11, 30, 30), (48, 11, 30, 29), (48, 11, 29, 48), (48, 10, 48, 48), (0, 32, 0, 4), (0, 31, 13, 4), (9, 32, 0, 3), (9, 31, 13, 3), (28, 32, 0, 2), (28, 31, 13, 2), (47, 32, 0, 1), (47, 31, 13, 1), (0, 31, 12, 13), (0, 31, 11, 32), (0, 30, 32, 4), (9, 30, 32, 3), (28, 30, 32, 2), (47, 30, 32, 1), (0, 30, 31, 13), (0, 30, 30, 32)}
6	{(0, 1, 0, 2), (0, 0, 4, 2), (13, 1, 0, 1), (13, 0, 4, 1), (0, 0, 3, 11), (13, 0, 3, 10), (0, 0, 2, 30), (13, 0, 2, 29), (13, 0, 1, 48), (12, 13, 0, 1), (12, 12, 13, 1), (12, 11, 32, 1), (11, 32, 0, 1), (11, 31, 13, 1), (11, 30, 32, 1), (32, 1, 0, 0), (32, 0, 4, 0), (32, 0, 3, 9), (32, 0, 2, 28), (32, 0, 1, 47), (31, 13, 0, 0), (31, 12, 13, 0), (31, 11, 32, 0), (30, 32, 0, 0), (30, 31, 13, 0), (30, 30, 32, 0)}
7	{(0, 1, 0, 1), (0, 0, 2, 1), (1, 1, 0, 0), (1, 0, 2, 0), (0, 4, 0, 0), (0, 3, 11, 0), (0, 2, 30, 0), (0, 0, 1, 13), (13, 0, 1, 12), (32, 0, 1, 11), (0, 0, 0, 32), (13, 0, 0, 31), (32, 0, 0, 30)}
8	{(1, 1, 0, 0), (1, 0, 1, 0), (0, 0, 0, 1), (4, 0, 0, 0), (0, 2, 0, 0), (0, 1, 13, 0), (0, 0, 32, 0)}
9	{(0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1), (2, 0, 0, 0)}
10	{(0,0,0,1), (1,0,0,0), (0,1,0,0), (0,0,1,0)}

3.5 Summary

In this chapter, we have presented the design of two lightweight cryptographic permutations sLiSCP and sLiSCP-light. We have provided an in-depth analysis of security properties of Simeck sbox using SAT/SMT and MILP. Finally, we discussed the security of both permutations with respect to differential and linear, and algebraic distinguishers. We have shown that sLiSCP-light is better in algebraic properties, but weaker in differential and linear properties than sLiSCP.

Chapter 4

ACE

Contents

Declaration of Contributions

This chapter is based on [7]. My main contributions are as follows.

- Overall design and parameters selection with rationale.
- Modeled the following properties.
  - MILP model to compute the minimum number of active Simeck sboxes. The security bounds against differential and linear distinguishers.
  - Bit-based division property MILP model to find algebraic degree bounds, integral and zero-sum distinguishers.
  - Diffusion behavior.

4.1 Motivation

A sponge-based hash function utilizing a b-bit permutation with $b = r + c$, r-bit rate and c-bit capacity can only provide collision security up to $\frac{c}{2}$ bits (by birthday bound) (Section 2.3.3.2). Thus, to achieve 128-bit collision security with 256-bit message digest, c should be 256. To meet this security level, we require a permutation where $b \geq 257$. In this context, we explore the design structures of sLiSCP and sLiSCP-light permutations with large state sizes, and aim to design a permutation that can achieve a balance between hardware cost and software efficiency for both hashing and AEAD functionalities. Furthermore, it should have a simple analysis and sufficient security margins against distinguishing attacks.

This leads to the ACE which is often considered as one of the strongest cards in a deck of cards. In our case, ACE is a 320-bit permutation and a generalization of sLiSCP and sLiSCP-light permutations with five 64-bit blocks. In this chapter, we present its design and security analysis.

Outline. The rest of the chapter is organized as follows. In Section 4.2, we present the complete specification of the ACE permutation. Sections 4.3 and 4.4 provide the detailed security analyis and the rationale of our design choices.

4.2 The ACE Permutation

ACE is an iterative permutation that takes a 320-bit state as an input and outputs a 320-bit state after iterating the step function ACE-step for $s = 16$ times (Figure 4.1). The nonlinear operation SB-64 is applied on even indexed words, i.e., A, C and E, and hence the permutation name. We present the algorithmic description of ACE in Algorithm 4.1 and describe the individual components as follows.

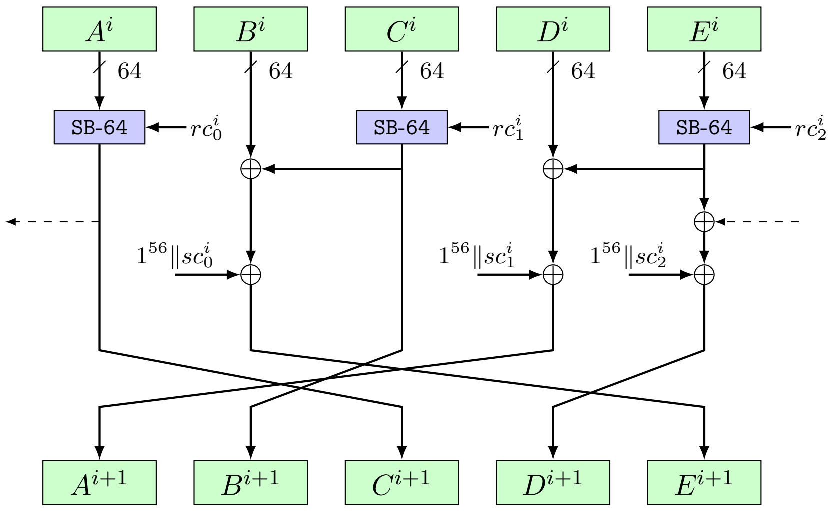


Figure 4.1: ACE-step

4.2.1 The nonlinear function SB-64

We use Simeck block cipher with block size 64 and $u = 8$ as the nonlinear operation. In particular, ACE utilizes SB-$[64, 8]$ (Section 3.2.1). In the following, we denote SB-$[64, 8]$ by

Algorithm 4.1 ACE permutation

```
1: Input: $S^0 = A^0||B^0||C^0||D^0||E^0$
2: Output: $S^{16} = A^{16}||B^{16}||C^{16}||D^{16}||E^{16}$

3: for $i = 0$ to 15 do:
4:         $S^{i+1} \leftarrow \mathsf{ACE\text{-}step}(S^i)$
5: return $S^{16}$

6: Function ACE-step($S^i$):
7:         $A^i \leftarrow \texttt{SB-64}(A_1^i||A_0^i, rc_0^i)$        ▷ $A_1^i, A_0^i$ are left and right halves of $A^i$
8:         $C^i \leftarrow \texttt{SB-64}(C_1^i||C_0^i, rc_1^i)$
9:         $E^i \leftarrow \texttt{SB-64}(E_1^i||E_0^i, rc_2^i)$
10:        $B^i \leftarrow B^i \oplus C^i \oplus (1^{56}||sc_0^i)$
11:        $D^i \leftarrow D^i \oplus E^i \oplus (1^{56}||sc_1^i)$
12:        $E^i \leftarrow E^i \oplus A^i \oplus (1^{56}||sc_2^i)$
13:        $A^{i+1} \leftarrow D^i$
14:        $B^{i+1} \leftarrow C^i$
15:        $C^{i+1} \leftarrow A^i$
16:        $D^{i+1} \leftarrow E^i$
17:        $E^{i+1} \leftarrow B^i$
18:        return $(A^{i+1}||B^{i+1}||C^{i+1}||D^{i+1}||E^{i+1})$

19: Function SB-64($x_1||x_0, rc$):
20:     $(q_7, q_6, \ldots, q_0) \leftarrow rc$
21:     for $j = 2$ to 9 do
22:         $x_j \leftarrow (\mathsf{L}^5(x_{j-1}) \odot x_{j-1}) \oplus \mathsf{L}^1(x_{j-1}) \oplus x_{j-2} \oplus (1^{31}||q_{j-2})$
23:     return $(x_9||x_8)$
```

`SB-64` if $u = 8$ is fixed.

4.2.2 Round and step constants

The step function of ACE is parameterized by two sets of triplets (rc_0^i, rc_1^i, rc_2^i) and (sc_0^i, sc_1^i, sc_2^i) where each rc_j^i and sc_j^i is of length 8 bits and $j = 0, 1, 2$. We call them round constants and step constants, respectively. As shown in Figure 4.1, the round constant triplet (rc_0^i, rc_1^i, rc_2^i) is used within the Simeck sboxes while the step constant (sc_0^i, sc_1^i, sc_2^i) is XORed to the words B, D and E. In Table 4.1 we list the hexadecimal values of the constants.

Table 4.1: Round and step constants of ACE

Step i	**Round constant** (rc_0^i, rc_1^i, rc_2^i)	**Step constant** (sc_0^i, sc_1^i, sc_2^i)
0 - 3	(07, 53, 43), (0a, 5d, e4), (9b, 49, 5e), (e0, 7f, cc)	(50, 28, 14), (5c, ae, 57), (91, 48, 24), (8d, c6, 63)
4 - 7	(d1, be, 32), (1a, 1d, 4e), (22, 28, 75), (f7, 6c, 25)	(53, a9, 54), (60, 30, 18), (68, 34, 9a), (e1, 70, 38)
8 - 11	(62, 82, fd), (96, 47, f9), (71, 6b, 76), (aa, 88, a0)	(f6, 7b, bd), (9d, ce, 67), (40, 20, 10), (4f, 27, 13)
12 - 15	(2b, dc, b0), (e9, 8b, 09), (cf, 59, 1e), (b7, c6, ad)	(be, 5f, 2f), (5b, ad, d6), (e9, 74, ba), (7f, 3f, 1f)

4.2.3 The linear layer

After XORing the blocks and step constants, five words are shuffled in a $(3, 2, 0, 4, 1)$ order (Lines 13-17 of Algorithm 4.1). This order of shuffling is different than the left cyclic shuffle of sLiSCP and sLiSCP-light (Figure 3.3).

4.3 Security Analysis

In this section, we analyze the security of ACE permutation by assessing its indistinguishability properties against various distinguishing attacks. We also compare the linear layer of ACE with other linear layers. In our analysis, we denote the linear layer by π, i.e., π permutates the blocks of state. For example, if $\pi(0, 1, 2, 3, 4) = (3, 2, 0, 4, 1)$, then after applying π, the state $A||B||C||D||E$ is transformed to $D||C||A||E||B$. Also, recall that u and s refer to the number of rounds inside the Simeck sbox and the number of steps, respectively. Moreover, by the component function f_j^s we refer to the ANF of the j-th bit of ACE after s steps.

4.3.1 Diffusion

Full bit diffusion. We say ACE achieves full bit diffusion after s steps if f_j^s is a function of all the input state bits for each $j \in \{0, \cdots, 319\}$. Thus, we need to find a minimum value of s which satisfies this criterion. The value depends on the diffusion property of the Simeck sbox as well. We find that $u = 11$ gives full bit diffusion within the Simeck sbox. Since ACE has five words that are updated in each step, we note that s has to be at least 5. Accordingly, we search for $(u, s) \in \{(i, 5)|1 \leq i \leq 11\}$. In Table 4.2, we list all the linear layers which achieve full bit diffusion in $(u, s) = (4, 5)$.

Table 4.2: Linear layers which achieve full bit diffusion in $(u, s) = (4, 5)$

No.	Linear layer π
1	(1, 2, 4, 0, 3)
2	(2, 0, 3, 4, 1)
3	(2, 0, 4, 1, 3)
4	(2, 4, 1, 0, 3)
5	**(3, 2, 0, 4, 1)**
6	(3, 2, 4, 0, 1)
7	(4, 0, 1, 2, 3)
8	(4, 2, 0, 1, 3)
9	(4, 2, 3, 0, 1)
10	(4, 3, 1, 0, 2)

Although $(u, s) = (4, 5)$ is the minimal choice set, we cannot choose it as we also need to ensure good resistance against differential and linear, and algebraic properties. For $u = 8$ and $s = 5$, the number of linear layers satisfying the full bit diffusion property are 13, and $\pi = (3, 2, 0, 4, 1)$ is one among them. We justify these choices later in Section 4.4.6.

Meet-in-the-middle distinguishers. Given that $(u, s) = (8, 16)$ and $\pi = (3, 2, 0, 4, 1)$ for ACE, we claim that such a distinguisher cannot cover more than ten steps, because ten steps guarantees full bit diffusion in both forward and backward directions.

4.3.2 Differential and linear cryptanalysis

MILP model for bounding minimum number of active sboxes. We model the difference propagation of ACE using MILP which is similar to the MILP model for sLiSCP (Section 3.4.1.1) with minor changes. The changes are listed below.

1. Model the difference propagation for 3 XORs.
2. Change $\pi = (1, 2, 3, 0)$ to $\pi = (3, 2, 0, 4, 1)$ for 5 blocks.
3. Change the objective function to incorporate the effect of 3 sboxes in one step.

Table 4.3 depicts the minimum number of active Simeck sboxes for ACE.

Table 4.3: Minimum number of active Simeck sboxes for s-step ACE

step (s)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
# active sboxes	0	1	2	3	4	6	7	8	10	11	13	14	15	16	18	19

Estimated bounds of differential and linear characteristics. Let p denote the MEDP of u-round Simeck sbox in $\log_2(\cdot)$ scale. An in-depth analysis of values of p has been provided in Section 3.3.1. For $(u, s) = (8, 16)$, we have $p = -15.8$ and the maximum estimated differential characteristic probability is given by $2^{19\times-15.8} \approx 2^{-300.2}$. The maximum estimated linear square correlation of a linear characteristic is computed analogously using $\gamma = -15.6$ and equals $2^{-296.4}$ where $\gamma = \text{MELHSC}(\texttt{SB-64})$ (Section 3.3.1.3).

4.3.3 Algebraic properties

We use the bit based division property model as described in Section 3.3.2.1 to analyze the algebraic properties of ACE. To find the algebraic degree bounds, it is enough to find upper bounds on the algebraic degrees of component functions $f_0^s, f_{32}^s, f_{64^s}, f_{96}^s, f_{128}^s, f_{160}^s, f_{192}^s, f_{224}^s,$ f_{256}^s and f_{288}^s. This is because for SB-64, the algebraic degree is 36 for the first 32 component

Table 4.4: Bounds on the algebraic degree of ACE. We give the lower and upper bounds when the MILP model does not converge.

	Component function									
steps (s)	f_0^s	f_{32}^s	f_{64}^s	f_{96}^s	f_{128}^s	f_{160}^s	f_{192}^s	f_{224}^s	f_{256}^s	f_{288}^s
1	36	27	36	27	36	27	36	27	36	27
2	92	83	63	62	92	83	92	83	63	62
3	126	125	119	117-120	239-247	235-245	236-249	233-248	119	118-120
4	240-247	238-246	241-248	242-247	306-312	303-311	304-313	304-311	241-248	241-247

functions while it is 27 for the remaining ones. Table 4.4 provides the bounds on the algebraic degree for these component functions.

Note that since the number of words in ACE is odd, due to slow diffusion the algebraic degrees are 63 and 62 for the component functions f_{64}^2 and f_{96}^2, respectively. A similar trend can be seen for the component functions f_{256}^2 and f_{288}^2. This non-uniformity in degree continues till step five, after which the degree is stabilized to 304-313 due to full bit diffusion. We expect that the degree reaches 319 in six steps. In Table 4.5, we list the integral distinguishers of ACE. Note that the positions of constant bits are chosen based on the degree of Simeck sbox.

Table 4.5: Integral distinguishers of ACE

Steps s	Input division property	Balanced component functions
8	$A^{32}\|\|C\|\|A^{287}$	$f_{64}^8 - f_{127}^8$, $f_{256}^8 - f_{319}^8$
	$A^{96}\|\|C\|\|A^{223}$	None
	$A^{160}\|\|C\|\|A^{159}$	None
	$A^{224}\|\|C\|\|A^{95}$	$f_{64}^8 - f_{127}^8$, $f_{256}^8 - f_{319}^8$
	$A^{288}\|\|C\|\|A^{31}$	None

4.3.4 Symmetric distinguishers

In Section 3.3.3, we have seen that Simeck sbox has the rotational invariant property. This property could propagate to multiple steps if constants are not added at each round and step. Thus, a proper choice of round constants is required. Below we list properties of the constants which ensure that each step function of ACE is distinct.

- For $0 \leq i \leq 15$, $sc_0^i \neq sc_1^i \neq sc_2^i$
- For $0 \leq i \leq 15$, $(rc_0^i, rc_1^i, rc_2^i) \neq (sc_0^i, sc_1^i, sc_2^i)$
- For $0 \leq i, j \leq 15$ and $i \neq j$, $(rc_0^i, rc_1^i, rc_2^i) \neq (rc_0^j, rc_1^j, rc_2^j)$
- For $0 \leq i, j \leq 15$ and $i \neq j$, $(sc_0^i, sc_1^i, sc_2^i) \neq (sc_0^j, sc_1^j, sc_2^j)$.

4.4 Design Rationale

In this section, we discuss the design choices of ACE.

4.4.1 State size

Our main objective as mentioned in Section 4.1 is to choose b that provides 128-bit security for both hash and AEAD. The immediate choices are $b = 288, 320$ and 384. In ACE, we choose $b = 320$ as it provides a good trade-off among hardware and software requirements, security and efficiency. With this choice of b, ACE can have implementations in a wide range of platforms. We discard the other state sizes for the following reasons.

- Considering the lightweight applications, 384-bit state is too heavy in hardware.
- 288 is not a multiple of 64, hence, we cannot efficiently use inbuilt 64-bit CPU instructions for software implementation.

4.4.2 Step function

We aim to build a 320-bit permutation, hence we could have used a 4-block sLiSCP-light with 80-bit Simeck sboxes. We discarded this choice for the following reasons.

- For $n > 32$ and the shift parameter set (a, b, c) with $a \neq b \neq c$, $a > b$ and $\gcd(a-b, n) = 1$, it is not practical to evaluate most of the cryptographic properties.
- A Simeck sbox with parameters $(a, b, c) = (5, 0, 1)$ and $2n = 80$ has weak differential and linear properties as $\gcd(5, 40) \neq 1$.
- A 80-bit based software implementation is not efficient.

Consequently, we decided to use a 5-block state with three Simeck sboxes and wrap around the linear mixing between words A and E. We also decided to XOR SB-64(A) with SB-64(E), instead of E to avoid the need for an extra temporary 64-bit register to store the initial value of E.

4.4.3 Nonlinear layer: Simeck sbox (SB-64)

- Simeck has a hardware friendly round function that consists of simple bitwise XOR, AND and cyclic shift operations.
- It is practical to evaluate the MEDP and MELHSC values of SB-64 which are $2^{-15.8}$ and $2^{-15.6}$, respectively (Section 3.3.1).
- SB-64 has an algebraic degree of 36 which enables us to provide guarantees against algebraic attacks.

4.4.4 Linear layer: $\pi = (3, 2, 0, 4, 1)$

The choice of a linear layer is crucial for proper mixing among the blocks, which in turn affects the differential and algebraic properties. Out of 5! possible permutations of blocks, 44 do not exhibit fixed points. Moreover, we found that iterating such permutations for multiple rounds achieves different differential and algebraic bounds. Accordingly, we searched their space to find the ones that offer the best diffusion and result in the minimum number of active Simeck sboxes in the smallest number of steps. Accordingly, we picked $\pi = (3, 2, 0, 4, 1)$ as our linear layer.

4.4.5 Round and step constants

- **Three 8-bit unique step constants** (sc_0^i, sc_1^i, sc_2^i). The 3-tuple constant value is unique across all steps, hence it destroys any symmetry between the steps of the permutation.

- **Three 8-bit unique round constants** (rc_0^i, rc_1^i, rc_2^i). One bit of each round constant is XORed with the state of the Simeck sbox in each round to destroy the preservation of rotational properties. Moreover, we append 31 ‘1’ bits to each one bit constant, which results in many inversions, and breaks the propagation of the rotational property in one step.

4.4.6 Number of rounds and steps

Our rationale for choosing the number of rounds u and number of steps s of ACE is based on achieving the best trade-off between security and efficiency. By security and efficiency, we mean the value of (u, s) for which ACE is indistiguishable from a random permutation and $u \times s$ is minimum. We now justify the choice of $(u, s) = (8, 16)$ for ACE.

1. Our first criterion is that s should be at least $3 \times m$ where m is the number of #steps needed to achieve full bit diffusion in the state. This choice is inspired from [69] and directly adds a 33% security margin against meet-in-the-middle distinguishers, as in $2m$ steps full bit diffusion is achieved in both forward and backward directions. Hence, $m = 5 \implies u \geq 4$ and $s \geq 15$ (Section 4.3.1). However, we found that we cannot choose $u = 4, \ldots, 7$, because we also aim to achieve good resistance against differential and linear cryptanalysis. Note that having a smaller number of rounds results in a weaker Simeck sbox.

2. The second criterion is to push the upper bound of the differential characteristic probability of ACE to below 2^{-320}. This value depends on the MEDP of u-round Simeck sbox

and the number of such active sboxes in s steps (denote by n_s). We have $n_{15} = 18$ and $n_{16} = 19$ (Table 4.3).

Table 4.6: Optimal differential characteristic probability p of u-round Simeck sbox and the corresponding differential characteristics bounds of ACE for $s = 15, 16$.

u	4	5	6	7	8	9
$\log_2(p)$	-6	-8	-12	-14	-18	-20
$n_{15} \times \log_2(p)$	-108	-144	-216	-252	-324	-360
$n_{16} \times \log_2(p)$	-114	-152	-228	-266	-342	-380

Table 4.6 depicts that $(u, s) \in \{(8, 15), (8, 16), (9, 15), (9, 16)\}$. However, if we consider the differential effect (Definition 2.3), then the differential probability is $2^{-15.8}$ when $u = 8$. Accordingly, we have

$$n_{15} \times -15.8 = 18 \times -15.8 = -284.4$$
$$n_{16} \times -15.8 = 19 \times -15.8 = -300.2$$

Thus, we ignore $(u, s) = (8, 15)$ and choose $(u, s) = (8, 16)$. The other two choices are discarded from the efficiency perspective as $u \times s = 135$ (resp. 144) when $(u, s) = (9, 15)$ (resp. (9,16)) compared to 128 iterations when $(u, s) = (8, 16)$.

4.5 Summary

In this chapter, we have presented the design of ACE which is a 320-bit permutation. It is a variant of generalized feistel structure with five branches, and uses eight round Simeck block cipher and $\pi = (3, 2, 0, 4, 1)$ as the nonlinear and linear layer, respectively. We analyzed its security by considering its indistinguishable behavior against diffusion, differential/linear, algebraic and symmetry based distinguishers. Finally, we justified the choice of each component along with its parameter set.

Chapter 5

WAGE

Contents

Declaration of Contributions

This chapter is based on [8]. My main contributions are as follows.

- Overall design and parameters selection with rationale (except the two 7-bit sboxes and selection of the underlying finite field $\mathbb{F}_{2^7}$ polynomial).
- MILP model to compute the minimum number of active sboxes and the security bounds against differential distinguishers.
- Analysis of diffusion behavior and algebraic degree.

5.1 Motivation

Designing a secure cryptographic primitive which can guarantee theoretical randomness properties is a challenging task. For instance, the eSTREAM finalist Grain [73] uses a combination of NLFSR and LFSR to obtain a lower bound on the period of the keystream sequence. In particular, it is proved that the period of Grain output sequence is a multiple of the period of LFSR [76]. On the other hand, another eSTREAM profile II submission WG [101] cipher adopts a single NLFSR for its key initialization phase which is switched to linear filter generator during the keystream generation phase (Section 2.5.4). Consequently, it provides proven randomness properties such as long period, balance, large and exact linear complexity, and ideal 2-level autocorrelation.

In the previous chapters, we have introduced sLiSCP, sLiSCP-light and ACE permutations, which according to our security analysis are indistinguishable from a random permutation. However, we cannot guarantee any randomness property similar to the WG. Thus, we aim to design a permutation which could offer theoretical randomness properties in addition to the classic AEAD and hash functionalities with a cheap hardware overhead. This chapter introduces WAGE, a 259-bit permutation based on the design of WG stream cipher which can be used as an original WG cipher by nullifying few operations in the control circuit.

Outline. The rest of the chapter is organized as follows. In Section 5.2, we present the specification of the WAGE permutation. Sections 5.3 and 5.4 provides the detailed security analyis and the rationale of our design choices.

5.2 Specification of WAGE

WAGE is a 259-bit permutation defined over an extension field of $\mathbb{F}_{2^7}$. The primitive polynomial is of degree 37 over $\mathbb{F}_{2^7}$ and given by

$$\ell(y) = y^{37} + y^{31} + y^{30} + y^{26} + y^{24} + y^{19} + y^{13} + y^{12} + y^{8} + y^{6} + \omega$$

where ω is a root of the defining primitive polynomial in $\mathbb{F}_{2^7}$ given by

$$f(x) = x^7 + x^3 + x^2 + x + 1.$$

The state is divided into 37 7-bit words and denoted by $S^i = (S^i_{36}, \cdots, S^i_0)$ at the beginning of the i-th round. The core components of the permutation include two WGP and four SB sboxes defined over $\mathbb{F}_{2^7}$, and 7-bit round constant (rc^i_0, rc^i_1) tuple.

5.2.1 State update function of WAGE

The state update function of WAGE denoted by WAGE-StateUpdate$(\cdot)$ (Figure 5.1) takes as input the current state S^i and round constant tuple (rc^i_0, rc^i_1), and updates the state in 3 steps as follows.

1. Compute the linear feedback fb.

$$fb = S^i_{31} \oplus S^i_{30} \oplus S^i_{26} \oplus S^i_{24} \oplus S^i_{19} \oplus S^i_{13} \oplus S^i_{12} \oplus S^i_{8} \oplus S^i_{6} \oplus (\omega \otimes S^i_0)$$

2. Apply WGP and SB sboxes, and add round constants.

$$
\begin{aligned}
S_5^i &\leftarrow S_5^i \oplus \mathsf{SB}(S_8^i)\\
S_{11}^i &\leftarrow S_{11}^i \oplus \mathsf{SB}(S_{15}^i)\\
S_{19}^i &\leftarrow S_{19}^i \oplus \mathsf{WGP}(S_{18}^i) \oplus rc_0^i\\
S_{24}^i &\leftarrow S_{24}^i \oplus \mathsf{SB}(S_{27}^i)\\
S_{30}^i &\leftarrow S_{30}^i \oplus \mathsf{SB}(S_{34}^i)\\
fb &\leftarrow fb \oplus \mathsf{WGP}(S_{36}^i) \oplus rc_1^i
\end{aligned}
$$

3. Shift the register and update the last word with the feedback value.

$$
\begin{aligned}
S_j^{i+1} &\leftarrow S_{j+1}^i, 0 \le j \le 35\\
S_{36}^{i+1} &\leftarrow fb
\end{aligned}
$$

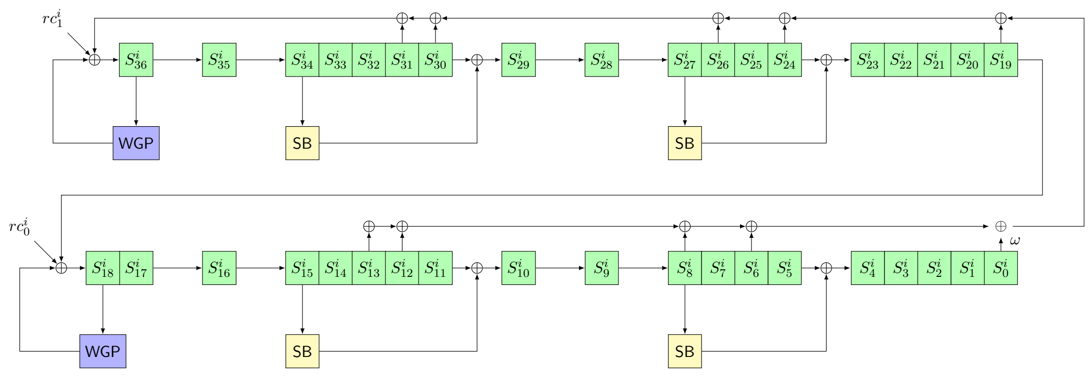


Figure 5.1: WAGE-StateUpdate function

5.2.2 Number of rounds and round constants

We update the state for 111 rounds with the WAGE-StateUpdate function. This value is chosen based on our security analysis (Section 5.3). The hex values of 111 round constant pairs are given in Appendix A.1.

5.2.3 WAGE to WG stream cipher

The WAGE permutation can be transformed to the WG stream cipher over $\mathbb{F}_{2^7}$ as follows.

1. Disconnect the second WGP module at position 18.

2. Disconnect all 4 SB modules at positions 8, 15, 27 and 34, and their corresponding XORs at positions 5, 11, 24, 30, respectively.

3. Force $rc_0^i = 0$ and $rc_1^i = 0$ for all i.

4. Add the Trace function module after the output of WGP at position 36.

For the applications where both AEAD and PRBG functionalities are required, we expect that the hardware overhead of going from WAGE to WG is much lower than two separate implementions.

Remark 5.1. We omit the mathematical details of the sboxes (treat them as a black box) as this is not a contribution of the thesis (see [8] for the exact details). However, we use their cryptographic properties (obtained from other designers[1] of [8]) to analyze the security of WAGE.

5.3 Security Analysis

In this section, we analyze the security of WAGE with respect to distinguishing attacks. We primarily focus on differential properties, diffusion behavior and algebraic properties.

5.3.1 Differential properties

Maximum differential probabilities of sboxes. In WAGE, we use two distinct 7-bit sboxes, namely WGP and SB as the nonlinear components. Since the word size is only 7 bits, we experimentally computed the maximun differential probabilities (MDP) of sboxes. The MDP values are given by

$$\mathrm{MDP}(\mathsf{WGP}) = 2^{-4.4}$$
$$\mathrm{MDP}(\mathsf{SB}) = 2^{-4}$$

Algorithmic description of MILP model. In Appendix A.2, we provide the exact Python code for constructing the optimization model. The model is generic for WAGE-like structures and takes as input $\vec{c}$ (the feedback tap positions), the position of sboxes and the number of rounds r. The output is $n_r(\vec{c})$ which is the minimum number of active sboxes. In Table 5.1, we list the values of $n_r(\vec{c})$ for varying $\vec{c}$ and $r \in \{37, 44, 51, 58, 74\}$.

Estimated bounds of differential characteristics. The maximum estimated differential characteristic probability (say p) of WAGE is given by

$$p = \max(2^{-4.42}, 2^{-4})^{n_r(\vec{c})} = 2^{-4\times n_r(\vec{c})}$$

For $r = 74$ and $\vec{c} = (31, 30, 26, 24, 19, 13, 12, 8, 6)$, we have $p = 2^{-4\times 59} = 2^{-236} > 2^{-259}$. Since Gurobi [2] is unable to finish for $r > 74$, we expect that for our choice of $\vec{c}$, $n_{111}(\vec{c}) \geq 65$.

[1] Kalikinkar Mandal

Table 5.1: Minimum number of active sboxes $n_r(\vec{c})$ for different primitive polynomials. Here '$-$' denotes that MILP optimization was too long and cannot be finished.

Primitive poly. coefficients	Rounds r				
$\vec{c}$	37	44	51	58	74
24, 23, 22, 21, 19, 6, 5, 4, 3	18	26	30	35	51
29, 27, 24, 23, 19, 11, 9, 6, 5	23	31	36	41	54
29, 28, 23, 22, 19, 11, 10, 5, 4	21	28	34	40	54
29, 28, 24, 20, 19, 11, 10, 6, 2	21	27	34	40	54
30, 28, 27, 21, 19, 12, 10, 9, 3	22	30	34	39	54
30, 29, 28, 26, 19, 12, 11, 10, 8	20	30	37	44	57
31, 25, 23, 21, 19, 13, 7, 5, 3	20	29	33	38	54
31, 26, 23, 20, 19, 13, 8, 5, 2	20	26	34	39	54
31, 28, 23, 21, 19, 13, 10, 5, 3	19	27	33	39	53
31, 30, 26, 24, 19, 13, 12, 8, 6	24	30	38	44	59
32, 25, 24, 21, 19, 14, 7, 6, 3	19	28	34	39	54
32, 29, 25, 22, 19, 14, 11, 7, 4	19	28	36	41	57
32, 29, 27, 22, 19, 14, 11, 9, 4	23	31	37	41	57
32, 29, 27, 24, 19, 14, 11, 9, 6	23	31	37	39	55
32, 30, 28, 24, 19, 14, 12, 10, 6	23	29	38	44	58
32, 31, 21, 20, 19, 14, 13, 3, 2	21	26	30	36	47
33, 27, 26, 20, 19, 15, 9, 8, 2	21	30	35	39	55
33, 29, 28, 21, 19, 15, 11, 10, 3	22	27	35	39	53
33, 30, 29, 26, 19, 15, 12, 11, 8	21	31	38	44	57
33, 31, 23, 22, 19, 15, 13, 5, 4	23	31	36	41	55
33, 31, 28, 23, 19, 15, 13, 10, 5	23	30	36	41	-
33, 31, 29, 22, 19, 15, 13, 11, 4	22	32	37	44	-
33, 31, 30, 25, 19, 15, 13, 12, 7	23	34	39	44	-

This is because for each additional 7 rounds, the number of active sboxes increases by at least 6 (row 10 in Table 5.1) which implies $p \leq 2^{-260} < 2^{-259}$.

5.3.2 Diffusion behavior

To achieve full bit diffusion, WAGE requires at least 21 rounds. This is because the 7 bits of S_{36} are shifted to S_0 in 21 clock cycles. Since the feedback function consists of 10 taps and all six sboxes (2 WGP and 4 SB) individually have the full bit diffusion property, WAGE achieves the full bit diffusion in at most 37 rounds. Accordingly, we claim that meet-in-the middle distinguishers may not cover more than 74 rounds as 74 rounds guarantee full bit diffusion in both the forward and backward directions.

5.3.3 Algebraic degree

The WGP and SB sboxes have an algebraic degree of 6. Note that if we only have WGP sbox at position S_{36} along with the feedback polynomial and exclude all other sboxes and intermediate XORs, then we get the original WG stream cipher. Such a stream cipher is resistant to attacks exploiting the algebraic degree if the nonlinear feedback used in the initialization phase is also used in the key generation phase (Chapter 8). Given that WAGE has 6 sboxes and we use nonlinear feedback for all of them, we expect that 111-round WAGE has high algebraic degree and is secure against integral and cube attacks.

5.4 Design Rationale

In this section, we justify our design choices that lead to WAGE. Our rationale is mainly based on security properties and hardware cost.

5.4.1 Finite field and state size

A trivial choice of finite field is $\mathbb{F}_{2^t}$ where $t \in \{5, 6, 7, 8\}$. We choose $\mathbb{F}_{2^7}$ and discard other choices based on the following.

- For $\mathbb{F}_{2^5}$ and $\mathbb{F}_{2^6}$, the state has many 5-bit and 6-bit words, respectively. Thus, it is not feasible to provide bounds for the differential distinguishers as the MILP model does not converge.

- The hardware cost[2] of WGP module over $\mathbb{F}_{2^8}$ is at least twice the cost of WGP module over $\mathbb{F}_{2^7}$.

Once the field size is set, we choose the state size to be a multiple of 7 and is at least 256 bits. Thus, we choose it as $37 \times 7 = 259$ bits.

5.4.2 Choice of sboxes

To preserve the structure of WG we require a WGP sbox at index 36. A single sbox results in a slower confusion and diffusion, and is not good from design perspective. Alternatively, we search for a 7-bit lightweight sbox whose differential and algebraic properties are similar to WGP. In order to keep the structure symmetric, we opt for even number of WGP and SB sboxes.

5.4.3 Positions and number of sboxes

The linear layer of WAGE is composed of 1) $\mathcal{L}_1$: a primitive polynomial of degree 37, which is primitive over $\mathbb{F}_{2^7}$ and 2) $\mathcal{L}_2$: input and output tap positions of WGP and SB sboxes. There exist many choices for $\mathcal{L}_1$ and $\mathcal{L}_2$, which results in a tradeoff between (especially) security and implementations. Note that we cannot have only $\mathcal{L}_1$ or only $\mathcal{L}_2$ as the linear layer, because

- only $\mathcal{L}_1$ results in a slower diffusion

[2]Obtained from the hardware group of [8]

- only $\mathcal{L}_2 \implies$ there are many words which are not mixed among themselves. Thus, the entire state can be divided into multiple independent sub-states. For example, the case when key bits are not mixed.

The required criteria for $\mathcal{L}_1$ and $\mathcal{L}_2$ are listed as follows.

1. To have a lightweight $\mathcal{L}_1$ we look for a primitive polynomial of the form

$$\ell(y) = y^{37} + \sum_{j=1}^{36} c_j y^j + \omega, \qquad c_j \in \mathbb{F}_2$$

   where ω is the root of the chosen field polynomial $f(x)$, which is also a primitive element of $\mathbb{F}_{2^7}$. Including ω, we choose feedback polynomials with few non-zero tap positions $(c_j = 1)$ which are symmetric.

2. A combination of $\mathcal{L}_1$ and $\mathcal{L}_2$ for which computing the minimum number of active sboxes is feasible and enable us to provide bounds for differential distinguishers.

Remark 5.2. Our MILP model does not converge when the number of sboxes is at least 8, and thus we cannot provide bounds for differential distinguishers for 8 or more sboxes. So we chose only six sboxes for WAGE, i.e., 2 WGP and 4 SB.

We analyze the 23 symmetric polynomials[3] (10 non-zero taps, Table 5.1) with respect to minimum number of active sboxes. We chose the one that provides the maximum resistance against differential attack. More precisely, we have

$$\begin{aligned} \mathcal{L}_1 &: \ y^{37} + y^{31} + y^{30} + y^{26} + y^{24} + y^{19} + y^{13} + y^{12} + y^{8} + y^{6} + \omega, \\ \mathcal{L}_2 &: \ \{(36,36), (34,30), (27,24), (18,19), (15,11), (8,5)\} \end{aligned}$$

where $(a, b) \in \mathcal{L}_2$ denotes the (input, output) position of an sbox (Figure 5.1).

5.4.4 Round constants

The structure of WAGE without round constants is symmetric and hence vulnerable to slide attacks. Thus, we XOR two 7-bit round constants rc_1^i and rc_0^i to words S_{36}^i and S_{18}^i, respectively. The round constant tuple is distinct for each round, i.e., $(rc_0^i, rc_1^i) \neq (rc_0^j, rc_1^j)$ for $0 \leq i, j \leq 110$ and $i \neq j$ ensuring that all the rounds of WAGE are distinct.

5.4.5 Number of rounds

Our rationale for selecting the number of rounds (say n_r) is to choose a value such that the WAGE permutation is indistinguishable from a random permutation. We justify our choice of

[3]Obtained from the hardware group of [8]

$n_r = 111$ as follows.

1. WAGE adopts an LFSR structure with 37 7-bit words, and hence $n_r \geq 37$, otherwise the words will not be mixed among themselves properly, which leads to meet-in-the-middle attacks.

2. For $n_r = 74$, the maximum estimated differential characteristic probability of WAGE equals $2^{-4\times 59} = 2^{-236} > 2^{-259}$. To push this value to less than 2^{-259}, $n_r \geq 74$. However, it is infeasible to compute the value for $n_r \geq 74$. Thus, we expect that for $n_r = 111$, the maximum estimated differential characteristic probability $\ll 2^{-259}$ (Section 5.3.1).

5.5 Summary

In this chapter, we have introduced WAGE which is a 259-bit permutation based on the WG stream cipher. To keep the overall design lightweight, we opt for a symmetric feedback polynomial with 10 non-zero taps and six 7-bit (2 WGP + 4 SB) sboxes. We analyzed the security of WAGE with respect to differential and algebraic distinguishers. Finally, we justified the choice of each component with their respective parameters.

Part II
Mode of Operations for AEAD and Hash

Chapter 6

Sponge Mode with a Unified Function

Contents

Declaration of Contributions

This chapter is based on [14, 16, 15, 7, 13, 8]. My main contributions are as follows.

- Equal contribution in design of the mode and parameters selection of AEAD and hash algorithms.
- Choice of domain separators, rate and capacity part of the state.
- Choice of positions of state for loading key and nonce bytes, and extracting tag bytes (except WAGE AEAD).

6.1 Motivation

The sponge framework is very diversified in terms of the offered functionalities such as AEAD, hash, MAC and PRBG (Section 2.3). Each functionality could be attained using the same cryptographic permutation P. The only differences arise from the implementation cost and security level we target. For instance, a hash digest of 256 bits restricts the size of permutation P to a minimum of 320 bits for a rate of 64 bits. The same permutation can also provide up to 128 bits authenticated encryption security with 192-bit rate. Although, the same permutation is used for both hash and AEAD, their hardware costs vary because of different rate values (64 XORs and 192 XORs). Thus, it is reasonable to have a circuitry which can provide multiple cryptographic functions with cheap overhead, which might be a determining factor for its realistic adoption in constrained devices. In this chapter, we introduce the design of the unified round function in a sponge framework which addresses the above problem.

Outline. The rest of the chapter is organized as follows. Section 6.2 presents the rationale and description of the unified round function. Sections 6.3 and 6.4 give a generic description of AEAD algorithm and their instances, respectively. We also discuss the choice and rationale of rate positions, domain separators, loading key and nonce procedures, and tag extraction procedures. In Section 6.5, we show how to handle short messages. Finally, we present hash algorithms in Section 6.6 and conclude in Section 6.7.

6.2 The Unified Round Function

In this section, we first discuss the need for a unified round function in a sponge framework and then give its explicit description.

6.2.1 Why a unified function?

We consider different scenarios which motivate us towards the design of a unified round function.

Security and hardware cost. In sponge-based keyed modes (Section 2.3.5.2), r-bit rate is known while c bits of capacity are unknown. An adversary with the knowledge of c bits can invert the permutation and hence recover the entire secret key. Thus, to prevent key recovery and forgery attacks from the knowledge of state, designs such as Ascon [60] mask the capacity with a key during the initialization and finalization phases (Figure 6.1). However, this masking requires an additional $|K|$ bits XORs and multiplexers. Additionally, these XORs and multiplexers are idle during associated data processing and encryption phases. Furthermore, to distinguish between different phases a single bit domain separator is only XORed to capacity after the last block of associated data is processed. This brings us to uniformity which is lacking here.

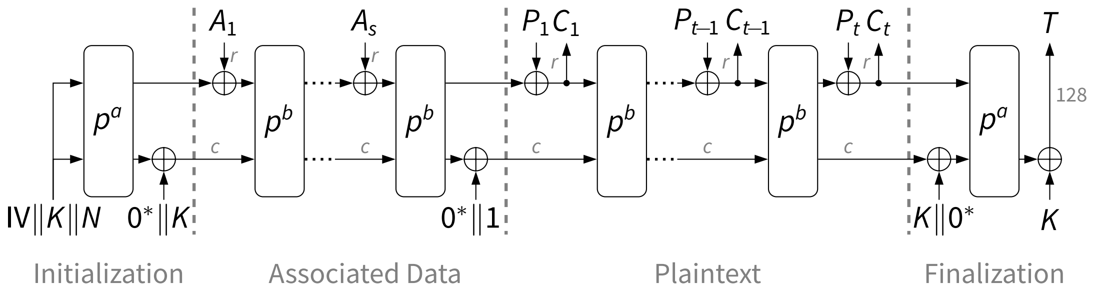


Figure 6.1: Duplex sponge mode for Ascon where p^a and p^b denote Ascon permutation with a and b rounds, respectively. Figure is taken from [60].

Uniformity. A simple way to have uniformity is to mask capacity with key and domain separators at each step. The former is not a good option from an efficiency perspective. Thus, we could use domain separators after each call of P (a technique similar to the one used in NORX [18]).

Our approach. We modify the keyed initialization and keyed finalization phases of the Ascon, and domain separation mechanism of NORX. In particular, instead of XORing key to capacity, we again absorb it in the state using the rate part for both phases. Moreover, we add a single bit domain separator after each call of the permutation. This approach makes key recovery hard even if the internal state is recovered and also brings uniformity across different phases. Additionally, the modification makes the initialization and finalization stages more hardware efficient and adaptable to different primitives. To this end, we only have one round function as described next.

6.2.2 Description of the unified round function

In Figure 6.2, we depict the unified round function which can be easily adaptable to different primitives. It incorporates absorption (X), squeezing (Y), domain separation, and according to the fed inputs, we decide which stage and functionality to implement (Table 6.1).

Some remarks.

- For decryption procedure, C is the input to the permutation.
- MAC is AEAD without encryption/decryption. PRBG is AEAD without associated data processing and encryption/decryption. So, both functionalities can be obtained from AEAD scheme only.

In the following, we describe AEAD and hash algorithms in detail.

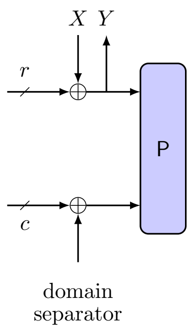


Figure 6.2: Schematic of the unified round function

Table 6.1: Different functionalities using the unified round function. Symbol '-' denotes no output.

Functionality	Phase	X	Y	Domain separator
AEAD	Initialization	K	-	`0x00`
	Associated data processing	AD	-	`0x01`
	Encryption/Decryption	M/C	C/M	`0x02`
	Finalization	K	-	`0x00`
Hash	Absorption	M	-	`0x00`
	Squeezing	-	r-bit digest	`0x00`
MAC	Initialization	K	-	`0x00`
	Message processing	M	-	`0x01`
	Finalization	K	-	`0x00`
PRBG	Initialization	K	-	`0x00`
	Output	-	r-bit random number	`0x00`

6.3 Generic AEAD Algorithm

In Algorithm 6.1, we present a high-level overview of an AEAD algorithm using the unified round function. We denote it by AE-[P] where P is the underlying permutation. Also, the b-bit ($b = r + c$) state of P is given by $S = (S_r, S_c)$ where S_r and S_c denote the rate and capacity part of the state, respectively. The encryption $\mathsf{AE_{enc}}$-[P] and decryption $\mathsf{AE_{dec}}$-[P] procedures of AE-[P] are shown in Figure 6.3. In the following, we first illustrate the padding rule and then describe each phase of AE-[P] in detail.

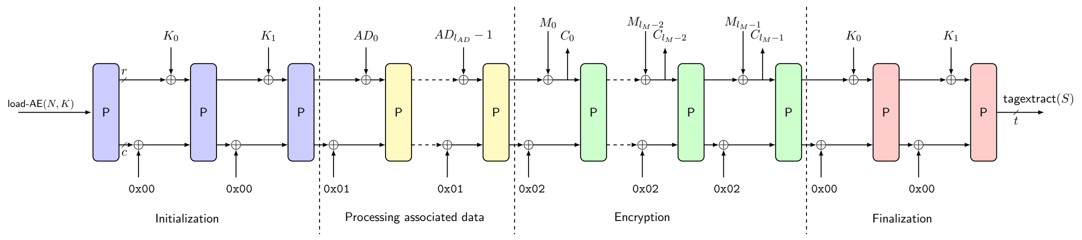
Algorithm 6.1 AEAD algorithm AE-[P]

```
1: Authenticated encryption $\mathsf{AE_{enc}}$-[P]$(K, N, AD, M)$:
2:     $S \leftarrow$ Initialization$(N, K)$
3:     if $|AD| \neq 0$ then:
4:         $S \leftarrow$ Processing-Associated-Data$(S, AD)$
5:     $(S, C) \leftarrow$ Encyption$(S, M)$
6:     $T \leftarrow$ Finalization$(S, K)$
7:     return $(C, T)$

8: Initialization$(N, K)$:
9:     $S \leftarrow$ load-AE$(N, K)$
10:    $S \leftarrow \mathsf{P}(S)$
11:    $(K_0||\cdots||K_{\ell_K - 1}) \xleftarrow{r} \mathsf{pad_r}(K)$
12:    for $i = 0$ to $\ell_K - 1$ do:
13:        $S \leftarrow (S_r \oplus K_i, S_c)$
14:        $S \leftarrow \mathsf{P}(S)$
15:    return $S$

16: Processing-Associated-Data$(S, AD)$:
17:    $(AD_0||\cdots||AD_{\ell_{AD}-1}) \xleftarrow{r} \mathsf{pad_r}(AD)$
18:    for $i = 0$ to $\ell_{AD} - 1$ do:
19:        $S \leftarrow (S_r \oplus AD_i, S_c \oplus 0^{c-2}||01)$
20:        $S \leftarrow \mathsf{P}(S)$
21:    return $S$

22: Encryption$(S, M)$:
23:    $(M_0||\cdots||M_{\ell_M - 1}) \xleftarrow{r} \mathsf{pad_r}(M)$
24:    for $i = 0$ to $\ell_M - 1$ do:
25:        $C_i \leftarrow M_i \oplus S_r$
26:        $S \leftarrow (C_i, S_c \oplus 0^{c-2}||10)$
27:        $S \leftarrow \mathsf{P}(S)$
28:    $C_{\ell_M - 1} \leftarrow$ trunc-msb$(C_{\ell_M - 1}, |M| \bmod r)$
29:    $C \leftarrow (C_0, C_1, \ldots, C_{\ell_M - 1})$
30:    return $(S, C)$

31: trunc-lsb$(X, n)$:
32:    return $(x_{r-n}, x_{r-n+1}, \ldots, x_{r-1})$
```

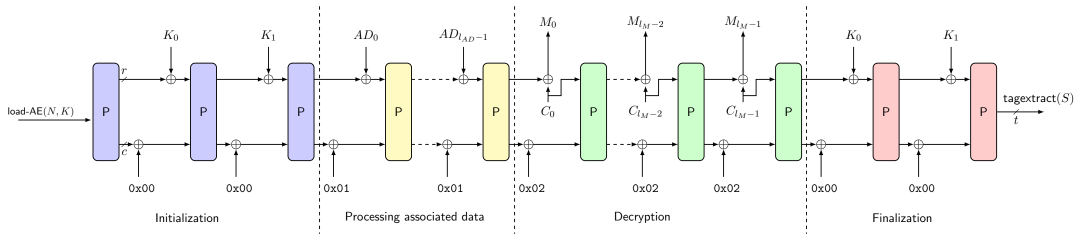
```
1: Verified decryption $\mathsf{AE_{dec}}$-[P]$(K, N, AD, C, T)$:
2:     $S \leftarrow$ Initialization$(N, K)$
3:     if $|AD| \neq 0$ then:
4:         $S \leftarrow$ Processing-Associated-Data$(S, AD)$
5:     $(S, M) \leftarrow$ Decyption$(S, C)$
6:     $T' \leftarrow$ Finalization$(S, K)$
7:     if $T' \neq T$ then:
8:         return $\perp$
9:     else:
10:        return $M$

11: Decryption$(S, C)$:
12:    for $i = 0$ to $\ell_C - 2$ do:
13:        $M_i \leftarrow C_i \oplus S_r$
14:        $S \leftarrow (C_i, S_c \oplus 0^{c-2}||10)$
15:        $S \leftarrow \mathsf{P}(S)$
16:    if $|C| \bmod r = 0$ then:
17:        $M_{\ell_C - 1} \leftarrow C_{\ell_C - 1} \oplus S_r$
18:        $S \leftarrow \mathsf{P}(C_{\ell_C - 1}, S_c \oplus 0^{c-2}||10)$
19:        $S \leftarrow \mathsf{P}(S_r \oplus 10^{r-1}, S_c \oplus 0^{c-2}||10)$
20:    else:
21:        $M_{\ell_C - 1} \leftarrow C_{\ell_C - 1} \oplus$ trunc-msb$(S_r, |C| \bmod r)$
22:        $C_{\ell_C - 1} \leftarrow C_{\ell_C - 1}||($trunc-lsb$(S_r, r - |C| \bmod r) \oplus 10^{r-|C| \bmod r})$
23:        $S \leftarrow \mathsf{P}(C_{\ell_C - 1}, S_c \oplus 0^{c-2}||10)$
24:    $M \leftarrow (M_0, M_1, \ldots, M_{\ell_C - 1})$
25:    return $(S, M)$

26: Finalization$(S, K)$:
27:    $(K_0||\cdots||K_{\ell_K - 1}) \xleftarrow{r} \mathsf{pad_r}(K)$
28:    for $i = 0$ to $\ell_K - 1$ do:
29:        $S \leftarrow \mathsf{P}(S_r \oplus K_i, S_c)$
30:    $T \leftarrow$ tagextract$(S)$
31:    return $T$

32: trunc-msb$(X, n)$:
33:    if $n = 0$ then:
34:        return $\phi$
35:    else:
36:        return $(x_0, x_1, \ldots, x_{n-1})$
```

6.3.1 Padding

Padding is required when the length of the processed data is not a multiple of the rate r value. The padding rule (10*), denoting a single 1 followed by the required number of 0's, is applied to the message M, so that its length after padding is a multiple of r. The resulting padded message is then divided into ℓ_M r-bit blocks $M_0\|\cdots\|M_{\ell_M-1}$. A similar procedure is carried out on the associated data AD which results in ℓ_{AD} r-bit blocks $AD_0\|\cdots\|AD_{\ell_{AD}-1}$. In the case where no associated data is present, no processing is necessary. For the secret key K, we simply append minimum number of 0's so that its length is a multiple of r. We summarize



(a) Authenticated encryption algorithm $\mathsf{AE}_{\mathsf{enc}}$-[P]



(b) Verified decryption algorithm $\mathsf{AE}_{\mathsf{dec}}$-[P]

Figure 6.3: AEAD algorithm AE-[P] in an unfied sponge mode for $2r$-bit key

the padding rules below.

$$\mathsf{pad_r}(K) = K \| 0^{r-(|K| \bmod r)}, \text{ if } |K| \bmod r \neq 0$$
$$\mathsf{pad_r}(M) = M \| 1 \| 0^{r-1-(|M| \bmod r)}$$
$$\mathsf{pad_r}(AD) = \begin{cases} \phi & \text{if } |AD| = 0 \\ AD \| 1 \| 0^{r-1-(|AD| \bmod r)} & \text{o.w.} \end{cases}$$

6.3.2 Domain separators

We use a lightweight domain separation mechanism where a different 2-bit constant (Table 6.2) is XORed to the capacity when a new phase starts. The domain separator could be XORed to any position in the capacity. We XOR it to the last 2 bits of the capacity for the sake of description[1]. In Table 6.3, we show that this domain separation mechanism along with the padding rule can distinguish all different types of processed data.

Table 6.2: Domain separation constants (in hex)

Initialization	Procesing AD	Encryption & Decryption	Finalization
0x00	0x01	0x02	0x00

[1]However, exact position depends on the choice of P.

Table 6.3: Domain separators sequence for different processed blocks

AD	M	Domain separators sequence	Processed blocks
Empty	Empty	`0x02`	10^{r-1}
	Partial	`0x02`	Partial M with padding
	Complete	`0x02, 0x02`	complete M block and 10^{r-1}
Partial	Empty	`0x01, 0x02`	Partial AD block with padding and 10^{r-1}
	Partial	`0x01, 0x02`	Partial AD and M blocks with padding
	Complete	`0x01, 0x02, 0x02`	Partial AD block with padding, complete M block and 10^{r-1}
Complete	Empty	`0x01, 0x01, 0x02`	complete AD block, 10^{r-1} and 10^{r-1}
	Partial	`0x01, 0x01, 0x02`	complete AD block, 10^{r-1} and partial M block with padding
	Complete	`0x01, 0x01, 0x02, 0x02`	complete AD block, 10^{r-1}, complete M block and 10^{r-1}

6.3.3 Initialization

The goal of this phase is to initialize the state S with an n-bit public nonce N and κ-bit secret key K. The state is first loaded using $\mathsf{load\text{-}AE}(N, K)$ (this function depends on the choice of P and is explicitly defined in Section 6.4.2). Afterwards, the permutation P is applied to the state, and the key blocks are absorbed into the state with P applied each time. The initialization steps are described below.

$$\begin{aligned} S &\leftarrow \mathsf{P}(\mathsf{load\text{-}AE}(N, K)) \\ S &\leftarrow \mathsf{P}(S_r \oplus K_i, S_c), \text{ for } i = 0 \text{ to } \ell_K - 1 \end{aligned}$$

6.3.4 Processing associated data

If there is associated data, each AD_i block, $i = 0, \ldots, \ell_{AD} - 1$ is XORed with the rate part of the state S, and the domain separator is XORed to capacity. Then, the permutation P is applied to the whole state.

$$S \leftarrow \mathsf{P}(S_r \oplus AD_i, S_c \oplus (0^{c-2} \| 01)),\ i = 0, \ldots, \ell_{AD} - 1$$

This phase is defined in Algorithm 6.1.

6.3.5 Encryption

Similar to the processing of associated data, however, with a different domain separator, each message block M_i, $i = 0, \ldots, \ell_M - 1$ is XORed to the S_r part of the state resulting in the corresponding ciphertext C_i. After the computation of C_i, the permutation P is applied to the state, i.e.,

$$\begin{aligned} C_i &\leftarrow S_r \oplus M_i, \\ S &\leftarrow \mathsf{P}(C_i, S_c \oplus (0^{c-2} \| 10)),\ i = 0, \cdots, \ell_M - 1 \end{aligned}$$

The last ciphertext block $C_{\ell_M - 1}$ is truncated so that its length is equal to that of the last unpadded message block. The details of encryption procedure are given in Algorithm 6.1.

The decryption procedure is symmetrical to the encryption procedure and is illustrated in Algorithm 6.1.

6.3.6 Finalization

After the extraction of the last ciphertext block, the domain separator is reset to `0x00` indicating the start of the finalization phase. Afterwards, the key blocks are absorbed into the state. Finally, the t-bit tag is extracted using the function $\mathsf{tagextract}(\cdot)$. The finalization steps are mentioned below and illustrated in Algorithm 6.1.

$$\begin{aligned} S &\leftarrow \mathsf{P}(S_r \oplus K_i, S_c),\ \text{for } i = 0 \text{ to } \ell_K - 1 \\ T &\leftarrow \mathsf{tagextract}(S). \end{aligned}$$

Note that the $\mathsf{tagextract}(\cdot)$ depends on the choice of P and is described later in Section 6.4.2.

6.3.7 On the security and data limit

The security proofs of sponge modes rely on the indistinguishability of the underlying permutation from a random one [31, 35, 34, 79]. In sponge based keyed modes, nonce reuse enables the encryption of two different messages with the same keystream, which undermines the privacy of the primitive. Moreover, the attacker can acquire multiple combinations of input and output differences which leak information about the capacity bits, and may lead to the construction of full state [35, 30]. On the other hand, a nonce reuse differential attack may be exploited if the attacker is able to inject a difference in the plaintext and cancel it out by another difference after the permutation call. However, such an attack depends on the probability of the best differential characteristic and the number of rounds of the underlying permutation. In summary, the sponge based AEAD schemes require a fresh nonce for each encryption.

We assume a nonce-respecting adversary, i.e., for a fixed K, the nonce N is never repeated for an encryption query. However, it could be repeated while querying the decryption oracle. Then to achieve κ-bit security with allowed data of 2^d bits, capacity should satisfy $c \geq \kappa + d + 1$ and $d \ll c/2$ [34]. Note that the actual effective capacity is $c - 2$ as 2 bits are used for domain separation. For instance, the parameters $c = 256, r = 64, \kappa = 128$ means there does not exist a better attack (to the best of our knowledge) than exhaustive search if $d \leq 125$. Recently, Jovanovic *et al.* [79] have shown that sponge based AEAD achieves higher security bound, i.e., $\min\{2^{b/2}, 2^c, 2^k\}$, compared to [34]. More precisely, following Jovanovic's bound, we could decrease the capacity to 128 and increase the rate by 3X ($r = 192$), and still achieve 128-bit security for $d \leq 64$. However, this requires an additional 128 XORs and cannot meet our objective to achieve both AEAD and hash functionalities (having the same security levels) using the unified round function. Nevertheless, this is another option with high throughput.

Remark 6.1. The nonce size n satisfies $n \leq \kappa$. The exact lengths for different instances are given in Table 6.4.

6.4 AEAD Instances

In this section, we present the concrete instantiations of AE-[P] using sLiSCP, sLiSCP-light, ACE and WAGE permutations. For a κ-bit key, the AEAD instance is denoted by AE-[P][κ]. We also give an explicit description of the load-AE($\cdot$) and tagextract($\cdot$) procedures along with the rate (S_r) and capacity (S_c) part of the state.

6.4.1 AEAD schemes and recommended parameters

Table 6.4 presents the AEAD instances along with their parameters and claimed security levels.

Some remarks.

- The integrity security includes the integrity of nonce, associated data and message.
- The original schemes (first 6 rows in Table 6.4) as described in [14, 15] use a 3-bit domain separation. This is not required as shown in Section 6.3.2.
- Spix [13] utilizes sLiSCP-light-256 in a monkey duplex mode [36]. More precisely, it uses sLiSCP-light-256 with 18 steps for intialization and finalization phases, and 9-step sLiSCP-light-256 for associated data processing and encryption phases.

Table 6.4: Recommended AEAD schemes

Algorithm	Parameters (in bits)						Security (in bits)		
	r	c	κ	n	t	$\log_2(d)$	Confidentiality	Integrity	Authenticity
AE-[sLiSCP-192][80]	32	160	80	80	80	72	80	80	80
AE-[sLiSCP-192][112]	32	160	112	80	112	40	112	112	112
AE-[sLiSCP-256][128]	64	192	128	128	128	56	128	128	128
AE-[sLiSCP-light-192][80]	32	160	80	80	80	72	80	80	80
AE-[sLiSCP-light-192][112]	32	160	112	80	112	40	112	112	112
AE-[sLiSCP-light-256][128]	64	192	128	128	128	56	128	128	128
AE-[ACE][128]	64	256	128	128	128	124	128	128	128
AE-[WAGE][128]	64	195	128	128	128	64	128	128	128
Spix	64	192	128	128	128	60	128	128	128

- All our proposed schemes have better or comparable performance relative to the state-of-the-art algorithms. The reader is referred to [14, 16, 15, 7, 13, 8] for details on hardware and software results.

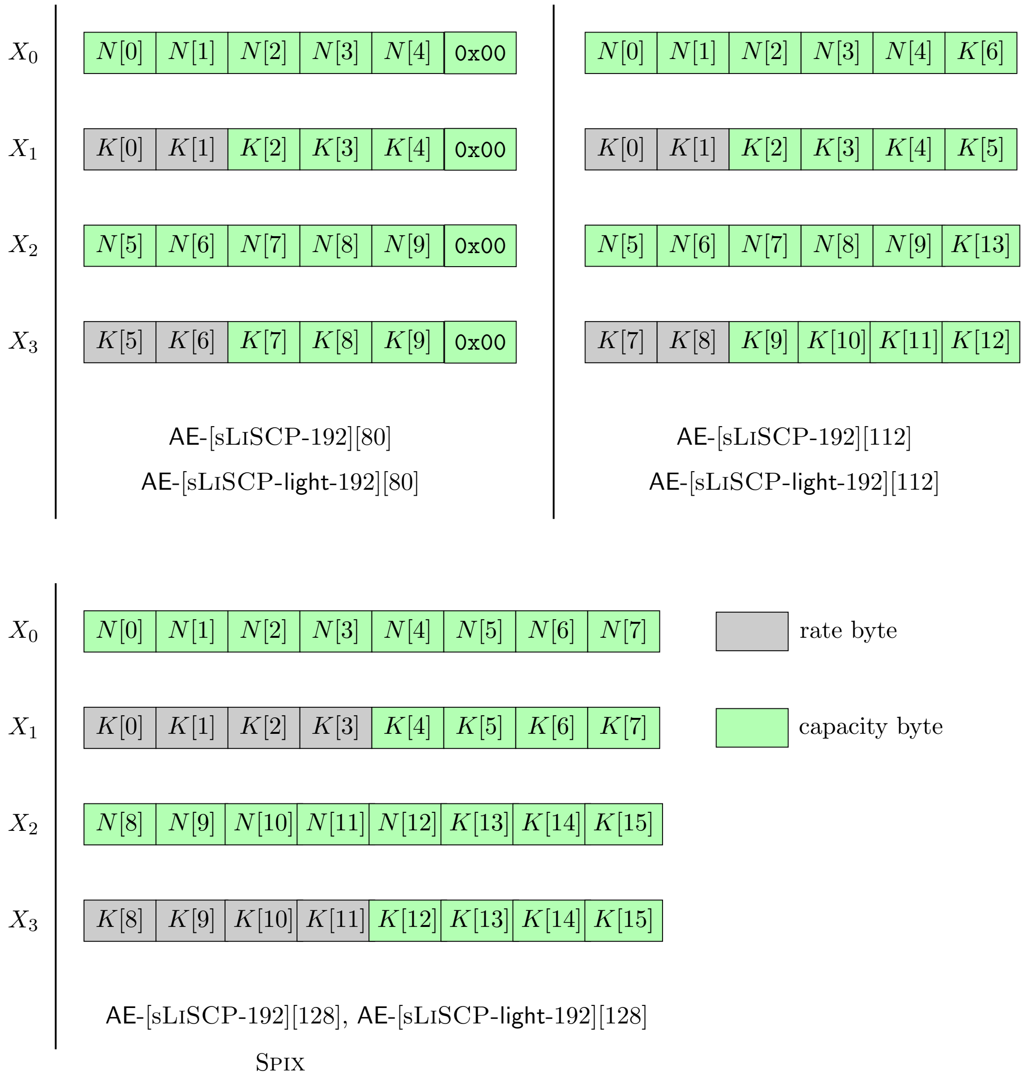
6.4.2 The load-AE and tagextract procedures and rate positions

Load-AE procedure (load-AE(N, K)). Figures 6.4-6.6 show the exact positions in the state for loading the key and nonce bytes where $K[i]$ and $N[i]$ denote the i-th byte of key and nonce starting from the left, respectively. In case of AE-[WAGE]-[128], we represent the loading procedure using bit notation.

Tagextract procedure (tagextract$(\cdot)$). For AE-[WAGE]-[128], the tag bits are extracted from the state bits where nonce is initialized (Figure 6.6). For all other AEAD instances, we extract tag bytes from the positions where the key bytes are loaded (Figures 6.4 and 6.5).

Rate and capacity. In Figures 6.4-6.6, the gray colored bytes/bits constitute the S_r part of state and are used for both absorbing the data (K, AD and M) and squeezing the ciphertext. The remaining green colored bytes/bits form the S_c part of state.

Rationale of rate positions. Our permutations follow a NLFSR based design paradigm. For the rate part, we want the input bits to be mixed properly as soon as possible so we achieve better confusion and diffusion. Accordingly, choosing the right place for absorbing the data determines how fast it is processed by the round function, which is important since not all the blocks in NLFSR-based constructions receive the same amount of processing. We also want to ensure that any injected difference in the rate part should activate as many



X_0 N[0] N[1] N[2] N[3] N[4] 0x00
X_1 K[0] K[1] K[2] K[3] K[4] 0x00
X_2 N[5] N[6] N[7] N[8] N[9] 0x00
X_3 K[5] K[6] K[7] K[8] K[9] 0x00
AE-[sLiSCP-192][80]
AE-[sLiSCP-light-192][80]
N[0] N[1] N[2] N[3] N[4] K[6]
K[0] K[1] K[2] K[3] K[4] K[5]
N[5] N[6] N[7] N[8] N[9] K[13]
K[7] K[8] K[9] K[10] K[11] K[12]
AE-[sLiSCP-192][112]
AE-[sLiSCP-light-192][112]
X_0 N[0] N[1] N[2] N[3] N[4] N[5] N[6] N[7]
X_1 K[0] K[1] K[2] K[3] K[4] K[5] K[6] K[7]
X_2 N[8] N[9] N[10] N[11] N[12] K[13] K[14] K[15]
X_3 K[8] K[9] K[10] K[11] K[12] K[13] K[14] K[15]
rate byte
capacity byte
AE-[sLiSCP-192][128], AE-[sLiSCP-light-192][128]
Spix

Figure 6.4: Visualization of load-AE(·) procedure of different AEAD instances

as sboxes as possible in the first few rounds to enhance resistance against differential and linear cryptanalysis. Furthermore, to decrease the probability of differential characteristics we choose the rate bytes/bits in a non-consecutive fashion.

A third party cryptanalysis [92] of AE-[sLiSCP-192] and AE-[sLiSCP-256] instances supports our rationale. Note that the authors in [92] could only attack 6/18 steps of these algorithms. Thus, we expect that the number of steps chosen for sLiSCP (18), sLiSCP-light (12) and ACE (16) provide a huge security margin against the best known attacks.

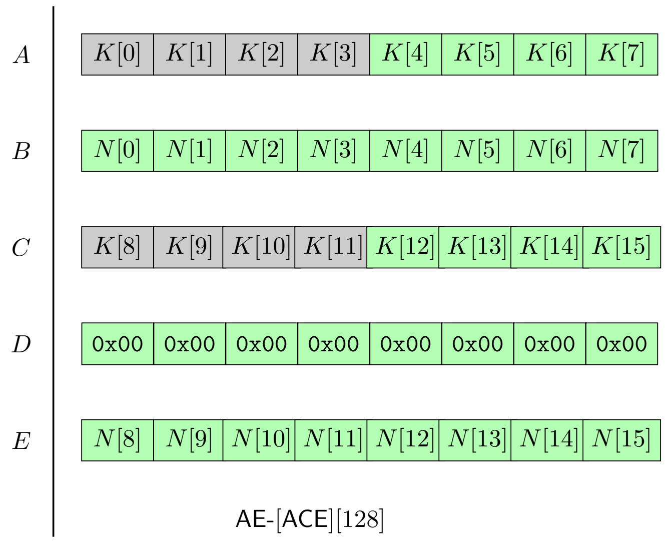

A
K[0] K[1] K[2] K[3] K[4] K[5] K[6] K[7]
B
N[0] N[1] N[2] N[3] N[4] N[5] N[6] N[7]
C
K[8] K[9] K[10] K[11] K[12] K[13] K[14] K[15]
D
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
E
N[8] N[9] N[10] N[11] N[12] N[13] N[14] N[15]
AE-[ACE][128]

Figure 6.5: Visualization of load-AE(·) procedure of ACE

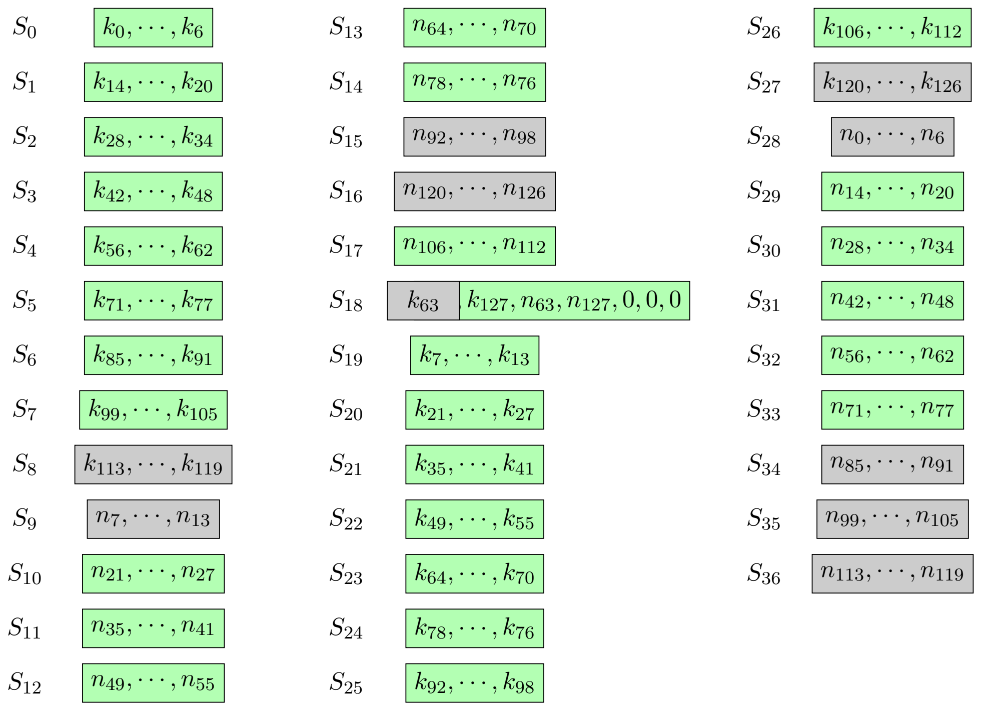

S_0 $k_0, \cdots, k_6$
S_1 $k_{14}, \cdots, k_{20}$
S_2 $k_{28}, \cdots, k_{34}$
S_3 $k_{42}, \cdots, k_{48}$
S_4 $k_{56}, \cdots, k_{62}$
S_5 $k_{71}, \cdots, k_{77}$
S_6 $k_{85}, \cdots, k_{91}$
S_7 $k_{99}, \cdots, k_{105}$
S_8 $k_{113}, \cdots, k_{119}$
S_9 $n_7, \cdots, n_{13}$
S_{10} $n_{21}, \cdots, n_{27}$
S_{11} $n_{35}, \cdots, n_{41}$
S_{12} $n_{49}, \cdots, n_{55}$
S_{13} $n_{64}, \cdots, n_{70}$
S_{14} $n_{78}, \cdots, n_{76}$
S_{15} $n_{92}, \cdots, n_{98}$
S_{16} $n_{120}, \cdots, n_{126}$
S_{17} $n_{106}, \cdots, n_{112}$
S_{18} k_{63} $k_{127}, n_{63}, n_{127}, 0, 0, 0$
S_{19} $k_7, \cdots, k_{13}$
S_{20} $k_{21}, \cdots, k_{27}$
S_{21} $k_{35}, \cdots, k_{41}$
S_{22} $k_{49}, \cdots, k_{55}$
S_{23} $k_{64}, \cdots, k_{70}$
S_{24} $k_{78}, \cdots, k_{76}$
S_{25} $k_{92}, \cdots, k_{98}$
S_{26} $k_{106}, \cdots, k_{112}$
S_{27} $k_{120}, \cdots, k_{126}$
S_{28} $n_0, \cdots, n_6$
S_{29} $n_{14}, \cdots, n_{20}$
S_{30} $n_{28}, \cdots, n_{34}$
S_{31} $n_{42}, \cdots, n_{48}$
S_{32} $n_{56}, \cdots, n_{62}$
S_{33} $n_{71}, \cdots, n_{77}$
S_{34} $n_{85}, \cdots, n_{91}$
S_{35} $n_{99}, \cdots, n_{105}$
S_{36} $n_{113}, \cdots, n_{119}$

Figure 6.6: Visualization of load-AE(·) procedure of WAGE

6.5 Handling Short Messages

The proposed AEAD mode in Section 6.3 requires $1 + \ell_{AD} + \ell_M + 2 \times \left\lceil \frac{|K|}{r} \right\rceil$ calls of P to generate the tag. For $|K| = 128, r = 64$ and short messages, say $\ell_{AD} = 1$ and $\ell_M = 1$, the number of calls of P is 7. We now describe two ways for handling short messages.

6.5.1 Remove key absorption layers

A naive way is to remove the key absorption layers from the initialization and finalization phases (Figure 6.3). This reduces the number of calls of P to $1 + \ell_{AD} + \ell_M$. *However, this approach affects the security (depending on the tag size) and proper care has to be taken while claiming security.* We illustrate this fact with an example. Consider the AEAD instance AE-[sLiSCP-192][112] (row 2 in Table 6.4). Our initial claim is 112-bit security. On removing the key absorption layers, the security drops to 80 bits. This is because after knowing the 112-bit tag the attacker guesses the remaining 80 state bits and then inverts the permutation to recover the master key.

6.5.2 Reduce number of rounds of P

While designing P and choosing its number of rounds, our approach was to ensure that P is indistinguishable from a random permutation. But when P is used in the unified sponge mode, an adversary can only control the r-bit rate part of the state. Thus, we could use reduced-round P for all phases instead of removing the key absorption layers. *Note that this approach require proper security analysis.*

Remark 6.2. We emphasize that there could be other ways of handling short messages as well. Here, we have presented the trivial ones only.

6.6 Generic Hash Algorithm and Instances

In Algorithm 6.2, we present a high-level overview of a hash algorithm using the unified round function. We denote it by Hash-[P] where P is the underlying permutation. Figure 6.3 shows an example of hash algorithm with message digest size $h = 4r$. We now describe each phase of Hash-[P] in detail.

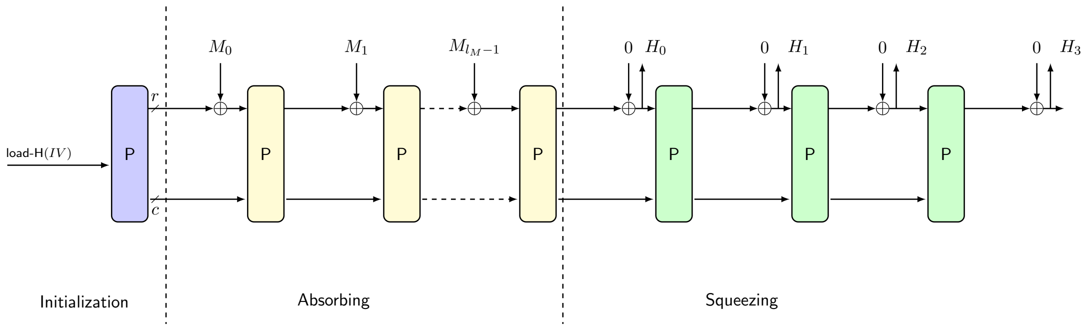


Figure 6.7: Hash algorithm Hash-[P] with $r = r'$ and $h = 4r$

Remark 6.3. Domain separation is not required for hash (equivalent to XORing `0x00` to capacity) as only one type of data is processed. The case of partial/complete message block can be distinguished by the padding rule described below.

6.6.1 Message padding

The padding rule (10*) similar to $\mathsf{AE\text{-}[P]}$ is applied to the input message M where a single 1 followed by 0's is appended to it such that its length after padding is a multiple of r. We denote the padding rule by

$$\mathsf{pad_r}(M) = M \| 10^{r-1-(|M| \bmod r)}$$

The resulting padded message is then divided into ℓ_M r-bit blocks $M_0 \| \cdots \| M_{\ell_M - 1}$.

6.6.2 Initialization

The state is first initialized with $IV = h/2 \| r \| r'$, where r/r' denotes the number of bits absorbed/squeezed per permutation call. Eight bits are used to encode each of the used $h/2$, r and r' sizes and loaded in the state[2] [70]. The remaining bytes are set to `0x00`. We denote this process by $\mathsf{load\text{-}H}$.

Remark 6.4. $IV = h/2 \| r \| r'$ is only used to distinguish different instances of hash using the same P. We could also start with all zero state for different P's.

After loading the state with the IV we call P once which completes the initialization phase.

$$S \leftarrow \mathsf{P}(\mathsf{load\text{-}H}(\mathsf{IV}))$$

6.6.3 Absorbing and squeezing

Each message block is absorbed by XORing it to the S_r part of the state, then P is applied. After absorbing all the message blocks, the h-bit output is extracted from the S_r part of the state r' bits (if $r' < r$ then we take the r' most significant bits of S_r) at a time followed by the application of P until a total of h/r' extractions are completed.

6.6.4 Security

For a sponge based hash with $b = r + c$ and h-bit message digest squeezed r' bits at a time, the generic security bounds [32, 70] are given by

[2]Exact positions are not required as IV is a constant value

Algorithm 6.2 Hash algorithm Hash-[P]

```
1: Hash-[P]($M, IV$):
2:     $S \leftarrow$ Initialization($IV$)
3:     $S \leftarrow$ Absorbing($S, M$)
4:     $H \leftarrow$ Squeezing($S$)
5:     return $H$

6: Initialization(IV):
7:     $S \leftarrow$ load-H($IV$)
8:     $S \leftarrow$ P($S$)
9:     return $S$

10: pad$_r$($M$) :
11:     $M \leftarrow M||10^{r-1-(|M| \bmod r)}$
12:     return $M$
```

```
1: Absorbing(S,M):
2:     $(M_0||\cdots||M_{\ell_M-1}) \leftarrow$ pad$_r$($M$)
3:     for $i = 0$ to $\ell_M - 1$ do:
4:         $S \leftarrow$ P($S_r \oplus M_i, S_c$)
5:     return $S$

6: Squeezing(S):
7:     for $i = 0$ to $h/r' - 1$ do:
8:       if $r = r'$ then:
9:           $H_i \leftarrow S_r$
10:      else:
11:          $H_i \leftarrow$ trunc-msb($S_r, r'$)
12:      $S \leftarrow$ P($S$)
13:    $H_{h/r'-1} \leftarrow S'_r$
14:    return $H_0||H_1||\cdots||H_{h/r'-1}$
```

- **Collision:** $\min(2^{h/2}, 2^{c/2})$
- **Preimage:** $\min(2^{\min(h,b)}, \max(2^{\min(h,b)-r'}, 2^{c/2}))$
- **Second-preimage:** $\min(2^{h}, 2^{c/2})$

6.6.5 Hash instances

Table 6.5 presents the hash instances along with their parameters and security levels. The rate part of the state is same as described in Figures 6.4 and 6.5.

Remark 6.5. The NIST LWC's requirement for the primary member of hash functions is at least 112-bit overall security with message digest size of 256 bits. This requirement can be met by using sLiSCP-256, sLiSCP-light-256 and WAGE permutations in hash mode with rate value 32 bits (last 3 rows of Table 6.5).

6.7 Summary

In this chapter, we have shown the construction of the unified round function in a sponge framework which is easily adaptable to multiple cryptographic primitives. As an application of it, we have presented generic AEAD and hash algorithms along with their concrete instantiations using sLiSCP, sLiSCP-light, ACE and WAGE permutations.

Table 6.5: Recommended hash instances

Algorithm	IV	h	r	r'	c	Preimage	Sec. Preimage	Collision
Hash-[sLISCP-192]	0x502020	160	32	32	160	128	80	80
Hash-[sLISCP-256]	0x604040	192	64	64	192	128	96	96
Hash-[sLISCP-256]	0x604020	192	64	32	192	160	96	96
Hash-[sLISCP-256]	0x604020	192	64	32	192	160	96	96
Hash-[sLISCP-light-192]	0x502020	160	32	32	160	128	80	80
Hash-[sLISCP-light-256]	0x604040	192	64	64	192	128	96	96
Hash-[sLISCP-light-256]	0x604020	192	64	32	192	160	96	96
Hash-[ACE]	0x804040	256	64	64	256	192	128	128
Hash-[sLISCP-256]	0x802020	256	32	32	224	224	112	112
Hash-[sLISCP-light-256]	0x802020	256	32	32	224	224	112	112
Hash-[WAGE]	0x802020	256	32	32	227	227	112	113

Chapter 7

SPOC: Sponge with Masked Capacity

Contents

Declaration of Contributions

This chapter is based on [12]. The design of SPOC mode comes from Ashwin Jha and Mridul Nandi of our team, while the design of SPOC's underlying permutation sLISCP-light is from this thesis. Accordingly, my main contributions are as follows.

- Design and analysis of sLISCP-light permutation as mentioned in Chapter 3.
- Choice of positions of state for rate, loading key and nonce bytes, and extracting tag bytes for SPOC.

7.1 Motivation

The best known bound for the sponge based AEAD in a single key setting is $\mathcal{O}(\frac{D^2+DT}{2^c})$ (Jovanic *et al.* [79]). Here D is the data complexity (the total amount of data which is authenticated and encrypted using distinct nonces) in bits, while T denotes the time complexity which includes the number of offline evaluations of the underlying permutation P. Note that the advantage term addresses the collisions probabilities, distinguishing advantage, verification query or the secret key recovery. While we have used the Bertoni *et al.* [34] bound in the previous chapter to ensure the same circuitry, all our proposed AEAD instances (Table 6.4) also satisfy the former bound. Now, for a 128-bit key, the NIST Lightweight Cryptography Project [97] has set the minimum values of D and T to be at least $2^{50} - 1$ bytes (2^{53} bits) and 2^{112}, respectively. For instance, consider sLiSCP-light-192 with $c = 160$ and $r = 32$. This does not meet NIST's requirement as $2^{53+112} > 2^{160}$. So, we could either increase the c value and decrease r, or increase the state size. Alternatively, is it possible to meet those requirements with state sizes less than 192? In this chapter, we present the design of Spoc which addresses the above problem. Spoc or Sponge with masked Capacity (pronounced as *Spock*) is a new AEAD mode of operation which offers higher security guarantee with smaller states. The underlying permutation of Spoc is sLiSCP-light permutation.

Outline. The rest of the chapter is organized as follows. Section 7.2 details the specification of Spoc and its recommended instances. We summarize the security claims of Spoc in Section 7.3 and then conclude in Section 7.4.

7.2 Specification of Spoc

In this section, we give a high level description of Spoc and then list its recommended instances. We also discuss the choice and rationale of rate positions, loading key and nonce procedures, and tag extraction procedures.

7.2.1 Spoc parameters

Spoc is primarily parameterized by rate r of the underlying permutation P where $r \in \{64, 128\}$. We write Spoc[r] to denote Spoc with the particular choice of rate value r. The secondary parameters are set as follows.

- Spoc[64]: In this version, $r = 64$, $c = 128$, $\kappa = 128$, $n = 128$ and $t = 64$.
- Spoc[128]: In this version, $r = 128$, $c = 128$, $\kappa = 128$, $n = 128$ and $t = 128$.

7.2.2 Description of Spoc

We denote the state of Spoc by $Y \| Z$ string where Y consists of c-bit capacity part of state while Z is made of r bits of rate. Figure 7.1 shows the outer level structure of Spoc. It takes as input X and XOR it to Y part of the state. If X is a plaintext/ciphertext block then only Z part of state is used as the keystream. It then XORs Z with a 4-bit domain separator d_X. An illustration of Spoc AEAD algorithm is shown in Figure 7.2 and the individual phases are explained below.

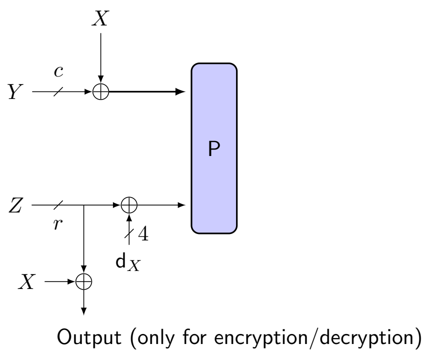


Figure 7.1: Schematic of SPOC outer layer

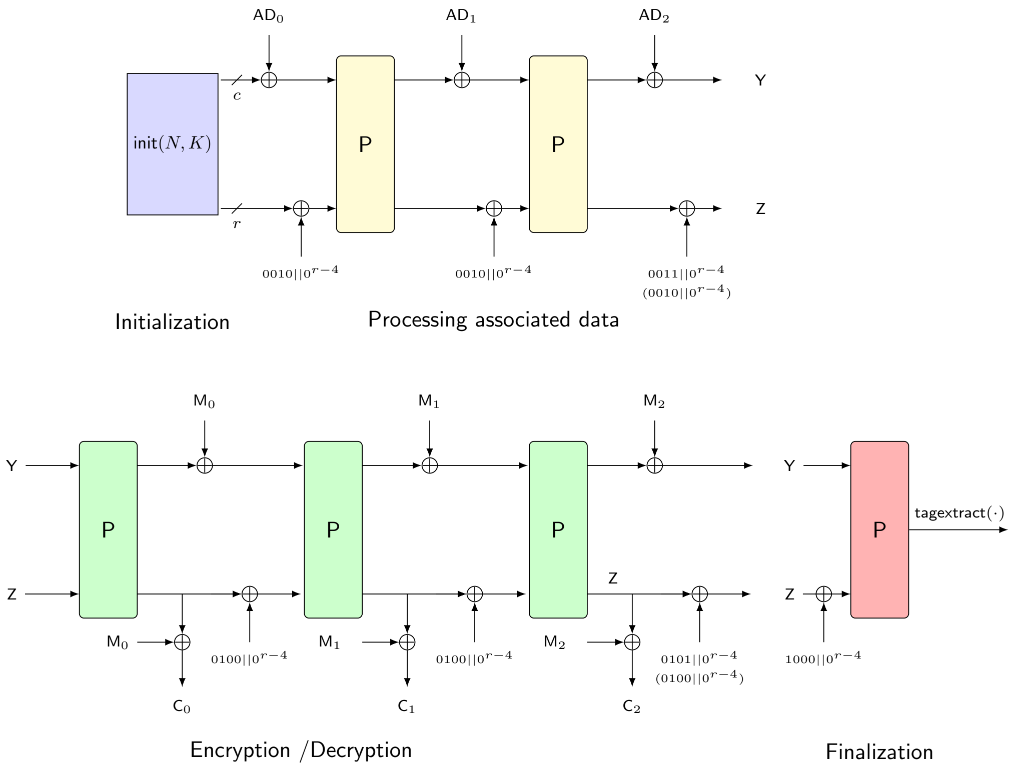


Figure 7.2: Schematic diagram of SPOC AEAD with 3 blocks of AD and M

7.2.2.1 Padding

The padding rule (10*), denoting a single 1 followed by the required number of 0's, is applied to data $X \in \{AD, M\}$, so that its length after padding is a multiple of r. The resulting padded data is then divided into ℓ_X r-bit blocks $X_0 \| \cdots \| X_{\ell_X - 1}$. In case of empty or complete blocks, no padding is required. Note that for SPOC-64, since $c = 128$ and $|X_i| = 64$, to match the capacity size we append 64 zeros to each X_i, i.e., $X_i \leftarrow X_i \| 0^{64}$.

7.2.2.2 Domain separators

Here we explain the 4-bit d_X that we use to separate the processing of various critical blocks. It is defined as $\mathsf{d}_X := \mathsf{ctrl}_{\mathsf{tag}}\,\mathsf{ctrl}_{\mathsf{pt}}\,\mathsf{ctrl}_{\mathsf{ad}}\,\mathsf{ctrl}_{\mathsf{par}}$. Initially, all the bits are set to 0. The bits are set to 1 in the following manner:

1. $\mathsf{ctrl}_{\mathsf{ad}}$: The bit sets to 1 during the processing of associated data blocks. For empty AD it remains set to 0.
2. $\mathsf{ctrl}_{\mathsf{pt}}$: The bit sets to 1 during the processing of plaintext blocks. For empty messages it remains set to 0.
3. $\mathsf{ctrl}_{\mathsf{par}}$: The bit sets to 1 at the last AD (M) block processing call if the last block is partial. For full last block it remains set to 0.
4. $\mathsf{ctrl}_{\mathsf{tag}}$: The bit sets to 1 at tag generation call.

We XOR d_X to the four most significant bits of Z. In case of encryption or decryption, this is done after the extraction of keystream bits. Table 7.1 enumerates all possible values for the domain separators along with their meanings.

Table 7.1: Possible values of d_X

d_X	**Meaning**
0000	Implicitly used in nonce processing
0010	Full AD block processing
0011	Partial AD block processing
0100	Full M/C block processing
0101	Partial M/C block processing
1000	Tag generation in empty AD and M/C case
1010	Tag generation in non-empty AD with full last block and empty M/C
1011	Tag generation in non-empty AD with partial last block and empty M/C
1100	Tag generation in (non-)empty AD and non-empty M/C with full last block
1101	Tag generation in (non-)empty AD and non-empty M/C with partial last block

7.2.2.3 Initialization

In this phase, we create the initial state using the nonce $N = N_0||N_1$ and the secret key $K = K_0||K_1$. We denote it by $\mathsf{init}(N, K)$ and is given by

$$\mathsf{init}(N,K) := Y||Z \leftarrow \begin{cases} \mathsf{P}(\mathsf{load}\text{-}\textsc{Spoc}[64](N_0,K)) \oplus (N_1||0^{128}) & \text{for } \textsc{Spoc}[64], \\ \mathsf{load}\text{-}\textsc{Spoc}[128](N,K) & \text{for } \textsc{Spoc}[128] \end{cases}$$

The function $\mathsf{load}\text{-}\textsc{Spoc}[r](\cdot)$ depends on the choice of P and assigns the nonce and key bytes to the particular byte positions of the state. We explicitly define this function in Section 7.2.5.

7.2.2.4 Processing associated data

If there is associated data, each AD_i block, $i = 0, \ldots, \ell_{AD} - 1$ is XORed to the capacity part of the state, and the domain separator is XORed to the rate. Then, the permutation P is applied to the whole state.

$$\begin{aligned} Y\|Z &\leftarrow \mathsf{P}((Y \oplus AD_i)\|(Z \oplus (0^{r-4}\|0010))),\ i = 0, \ldots, \ell_{AD} - 2 \\ Y\|Z &\leftarrow \begin{cases} \mathsf{P}((Y \oplus AD_{\ell_{AD}-1})\|(Z \oplus (0^{r-4}\|0010))) & \text{for complete block} \\ \mathsf{P}((Y \oplus AD_{\ell_{AD}-1})\|(Z \oplus (0^{r-4}\|0011))) & \text{for partial block} \end{cases} \end{aligned}$$

7.2.2.5 Encryption and decryption

During this phase, the message block is XORed to the Y part of state while keystream bits are taken from the Z part. The encryption procedure is given by

$$\begin{aligned} C_i &= M_i \oplus Z,\ \text{and} \\ Y\|Z &\leftarrow \mathsf{P}((Y \oplus M_i)\|(Z \oplus (0^{r-4}\|0100))),\ i = 0, \ldots, \ell_M - 2 \\ C_{\ell_M-2} &= M_{\ell_M-1} \oplus Z \\ Y\|Z &\leftarrow \begin{cases} (Y \oplus M_{\ell_M-1})\|(Z \oplus (0^{r-4}\|0100)) & \text{for complete block} \\ (Y \oplus M_{\ell_M-1})\|(Z \oplus (0^{r-4}\|0101)) & \text{for partial block} \end{cases} \end{aligned}$$

The last ciphertext block C_{ℓ_M-1} is truncated so that its length is equal to that of the last unpadded message block. For the decryption, we first compute $M_i = Z \oplus C_i$ and then XOR M_i to the capacity part of the state.

7.2.2.6 Finalization

This phase is responsible for tag generation. At the tag generation call, the control signal is of the form $1xyz$ where the 3 least significant bits xyz depend on the previous processed data blocks. We denote the process of extracting tag from state by tagextract-Spoc$[r]$ and is given by

$$\textsf{tagextract-}\textsc{Spoc}[r] \leftarrow \mathsf{P}(Y \| (Z \oplus (0^{r-4} \| 1000))).$$

The exact description of tagextract-Spoc$[r]$ function is provided in Section 7.2.6.

7.2.3 Recommended instantiations

We instantiate Spoc with sLiSCP-light permutation (Chapter 3) to provide two lightweight AEAD instances which offer 112-bit security. The sLiSCP-light permutation is chosen for its well-analyzed structure and low hardware implementation cost. Table 7.2 presents the recommended parameter sets for two lightweight instances of Spoc.

Table 7.2: Recommended parameter sets of Spoc

Instance	b	r	κ	n	t	Data (in bytes)
Spoc[64]_sLiSCP-light[192]	192	64	128	128	64	2^{50}
Spoc[128]_sLiSCP-light[256]	256	128	128	128	128	2^{50}

Remark 7.1. The two Spoc algorithms are secure while the prescribed data (2^{50} bytes) and time limit of 2^{112} are respected.

7.2.4 Positions of rate and capacity

Figure 7.3 depicts the exact positions of the state which are used for r-bit keystream and for masking r-bits of capacity. It also shows the 1-1 correspondence between the state representation $Y \| Z$ of Spoc and $X_0 \| X_1 \| X_2 \| X_3$ of sLiSCP-light. Each block is given in bytes notation, e.g., $X_0 = X_0[0] \| X_0[1] \| \cdots \| X_0[5]$ for sLiSCP-light-192.

Rationale. The choice of rate and capacity positions for Spoc depends on the underlying permutation. Since we instantiate Spoc with sLiSCP-light permutation, we have followed a similar strategy in choosing the rate and capacity positions as the one that has been used in sLiSCP and sLiSCP-light (Section 6.4.2).

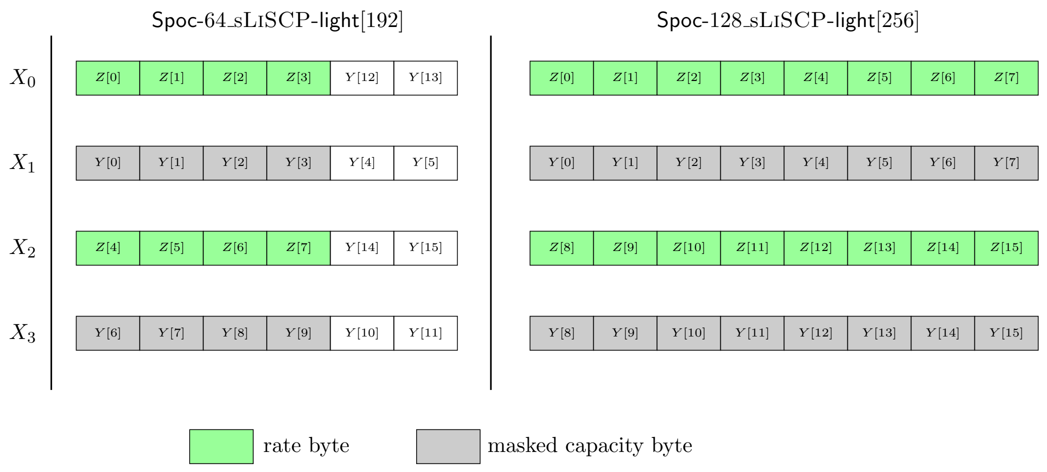

Spoc-64_sLiSCP-light[192]
Spoc-128_sLiSCP-light[256]
X_0
X_1
X_2
X_3
$Z[0]$ $Z[1]$ $Z[2]$ $Z[3]$ $Y[12]$ $Y[13]$
$Y[0]$ $Y[1]$ $Y[2]$ $Y[3]$ $Y[4]$ $Y[5]$
$Z[4]$ $Z[5]$ $Z[6]$ $Z[7]$ $Y[14]$ $Y[15]$
$Y[6]$ $Y[7]$ $Y[8]$ $Y[9]$ $Y[10]$ $Y[11]$
$Z[0]$ $Z[1]$ $Z[2]$ $Z[3]$ $Z[4]$ $Z[5]$ $Z[6]$ $Z[7]$
$Y[0]$ $Y[1]$ $Y[2]$ $Y[3]$ $Y[4]$ $Y[5]$ $Y[6]$ $Y[7]$
$Z[8]$ $Z[9]$ $Z[10]$ $Z[11]$ $Z[12]$ $Z[13]$ $Z[14]$ $Z[15]$
$Y[8]$ $Y[9]$ $Y[10]$ $Y[11]$ $Y[12]$ $Y[13]$ $Y[14]$ $Y[15]$
rate byte
masked capacity byte

Figure 7.3: Rate and capacity part of Spoc

7.2.5 Loading key and nonce

Here we describe the postions where the 128-bit key $K = K_0||K_1$ and 128-bit nonce $N = N_0||N_1$ is loaded in the state. In particular, we define the functions load-Spoc$[128](N, K)$ and load-Spoc$[64](N_0, K)$ of the init(N, K) procedure (Section 7.2.2.3).

load-SPOC$[128](N, K)$**.** For $0 \leq j \leq 7$, the state is loaded as follows.

$$X_1[j] \leftarrow K_0[j]$$
$$X_3[j] \leftarrow K_1[j]$$
$$X_0[j] \leftarrow N_0[j]$$
$$X_2[j] \leftarrow N_1[j]$$

load-SPOC$[64](N_0, K)$**.** The load-Spoc$[64](N_0, K)$ function is given by

$$X_1[0], \cdots, X_1[5] \leftarrow K_0[0], \cdots, K_0[5]$$
$$X_3[0], \cdots, X_3[5] \leftarrow K_1[0] \cdots, K_1[5]$$
$$X_0[0] \cdots, X_0[3] \leftarrow N_0[0], \cdots, N_0[3]$$
$$X_2[0], \cdots, X_2[3] \leftarrow N_0[4], \cdots, N_0[7]$$
$$X_0[4], X_0[5] \leftarrow K_0[6], K_0[7]$$
$$X_2[4], X_2[5] \leftarrow K_1[6], K_1[7]$$

7.2.6 Tag extraction

For Spoc[128]_sLiSCP-light[256], the tagextract-Spoc[128] function computes the 128-bit tag $T = T_0||T_1$ which is given by $T_0 \leftarrow X_1$ and $T_1 \leftarrow X_3$. Similarly, tagextract-Spoc[64] computes the 64-bit tag T of Spoc[64]_sLiSCP-light[192] as follows.

$$T[0], \cdots, T[3] \leftarrow S_1[0], \cdots, S_1[3]$$
$$T[4], \cdots, T[7] \leftarrow S_3[0], \cdots, S_3[3]$$

7.3 Security of Spoc Instances

In Table 7.3, we list the security levels of two instances of Spoc in a nonce-respecting setting. The numbers are based on our security analysis of sLiSCP-light permutation (Section 3.4) which is modeled as a random permutation for 18 steps. Accordingly, we do not claim security for Spoc with reduced-round sLiSCP-light permutation.

Table 7.3: Security levels of AEAD algorithms based on Spoc

AEAD algorithm	**Confidentiality**		**Integrity**	
	Time	Data (in bytes)	Time	Data (in bytes)
Spoc[64]_sLiSCP-light[192]	2^{112}	2^{50}	2^{112}	2^{50}
Spoc[128]_sLiSCP-light[256]	2^{112}	2^{50}	2^{112}	2^{50}

Remark 7.2. For a security proof of Spoc[1], the reader is referred to our NIST LWC round 2 candidate [12].

7.4 Summary

In this chapter, we have presented the design of Spoc, a new permutation based mode of operation for AEAD functionality. It satisfies NIST LWC AEAD requirements with 192-bit state when instantiated with sLiSCP-light-192.

[1]Not a contribution of this thesis

Part III

Cryptanalysis of Lightweight Symmetric Key Primitives

Chapter 8

MILP-based Cube Attack on the Reduced-Round WG-5 Lightweight Stream Cipher

Contents

Declaration of Contributions

This chapter is based on [115]. My main contributions are as follows.

- Modeled the division property based MILP models and key recovery attack on WG-5.
- Analysed and compared the design choices of WG-5 with Grain-128a and Trivium with respect to cube attacks.

8.1 Introduction

The cube attack is a powerful cryptanalytic technique for the analysis of stream ciphers (Section 2.4.4.2). Given the complicated algebraic normal form of keystream bits, conventional cube attacks always regard them as blackbox functions, and the attack is only feasible for smaller dimensional cubes. In Crypto 2017, Todo *et al.* [127] proposed cube attacks in a non-blackbox polynomial setting employing the division property [126]. Their technique takes the polynomial structure of the stream cipher into consideration by tracing the propagation of the division property through the initialization rounds. Accordingly, a theoretical proven upper bound on the number of secret key bits involved in the superpoly (Example in Section 2.4.4.2)and the complexity of its recovery is obtained. Consequently, they gave the best known key recovery attacks on reduced-round variants of Grain [11], Trivium [53] and Acorn [133].

In this chapter, we investigate the security of the key initialization phase of WG-5 [9] with respect to cube attacks in a non-blackbox polynomial setting using the division property. WG-5 is a lightweight version of the WG family of stream ciphers. The best cryptanalytic result on WG-5 is a univariate algebraic attack over $\mathbb{F}_{2^5}$ that recovers the 80-bit secret key using 2^{15} keystream bits in 2^{33} time [118]. Such an attack is applicable only when WG-5 runs a linear feedback keystream generation phase. Thus, analyzing the nonlinear feedback based initialization phase of WG-5 can provide better understanding of its security compared with Grain and Trivium.

Outline. The rest of the chapter is organized as follows. Sections 8.2 and 8.3 detail the specification of WG-5 and the attack details along with an algorithmic descriptions of all MILP models, respectively. In Section 8.4, we analyze the design parameters of WG-5, Grain-128a and Trivium with respect to cube attacks.

8.2 Specification of WG-5 Stream Cipher

WG-5 is a 160-bit stream cipher defined over an extension field of $\mathbb{F}_{2^5}$ (Figure 8.1). The 32 stage LFSR is defined using the primitive polynomial $x^{32}+x^7+x^6+x^4+x^3+x^2+\gamma$ where $\gamma = \alpha^4+\alpha^3+\alpha^2+\alpha+1$, and α is a root of field ($\mathbb{F}_{2^5}$) defining polynomial $x^5+x^4+x^2+x+1$. We denote the state of WG-5 at the beginning of the i-th round by $S^i = S^i[0]||S^i[1]||\ldots||S^i[31]$ where $S^i[j] = (s^i_{5j}, s^i_{5j+1}, s^i_{5j+2}, s^i_{5j+3}, s^i_{5j+4})$. The 80-bit secret key $k_0, k_1, \ldots, k_{79}$ and 80-bit initialization vector $v_0, v_1, \ldots, v_{79}$ are denoted by $K[0]||K[1]||\ldots||K[15]$ and $IV[0]||IV[1]||\ldots||IV[15]$, respectively. The cipher runs in two phases: key initialization phase and keystream generation (KSG) phase, which are explained below.

8.2.1 Key initialization phase

Initially, the state is loaded with K and IV as follows.

$$S^0[j] = \begin{cases} K[j \bmod 2], & \text{if } j \equiv 0 \bmod 2 \\ IV[j \bmod 2], & \text{o.w.} \end{cases}$$

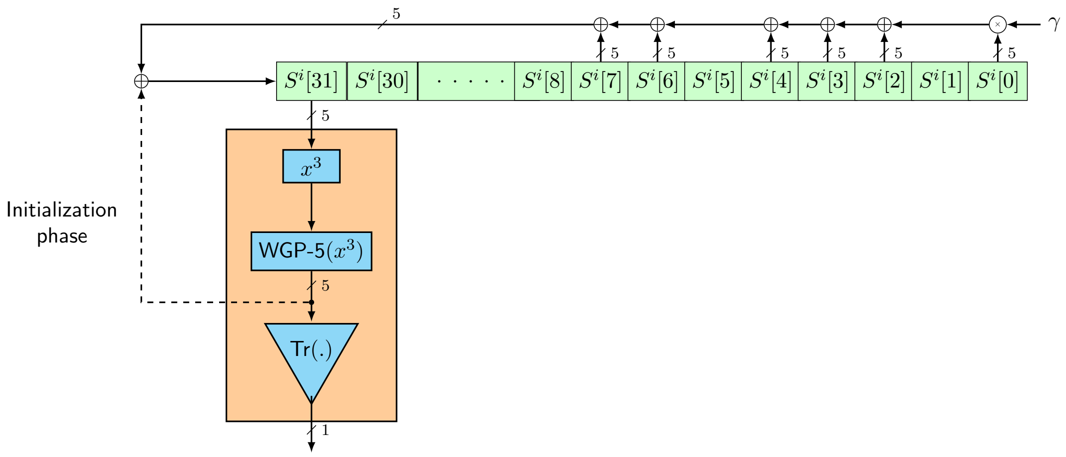

5
γ
$S^i[31]$ $S^i[30]$ · · · · · $S^i[8]$ $S^i[7]$ $S^i[6]$ $S^i[5]$ $S^i[4]$ $S^i[3]$ $S^i[2]$ $S^i[1]$ $S^i[0]$
Initialization phase
x^3
WGP-5(x^3)
Tr(.)
1

Figure 8.1: WG-5 stream cipher

The state is then updated for 64 rounds with the output of WGP-5 permutation feedback into the state, i.e., for $0 \leq i \leq 63$

$$\begin{aligned} fb &\leftarrow \gamma \otimes S^i[0] \oplus S^i[2] \oplus S^i[3] \oplus S^i[4] \oplus S^i[6] \oplus S^i[7] \oplus \mathsf{WGP\text{-}5}((S^i[31])^3) \\ S^{i+1}[j] &\leftarrow S^i[j+1], 0 \leq j \leq 30 \\ S^{i+1}[31] &\leftarrow fb \end{aligned}$$

8.2.2 Key generation phase

During the keystream generation phase, the keystream bit is computed by applying the Trace function Tr(·) function (Section 2.5.4) on the output of WGP-5 permutation. The state is then updated linearly without the feedback of WGP-5. More precisely, for $i \geq 64$

$$\begin{aligned} z_{i-64} &\leftarrow \mathsf{Tr}(\mathsf{WGP\text{-}5}(S^i[31])) \\ fb &\leftarrow \gamma \otimes S^i[0] \oplus S^i[2] \oplus S^i[3] \oplus S^i[4] \oplus S^i[6] \oplus S^i[7] \\ S^{i+1}[j] &\leftarrow S^i[j+1], 0 \leq j \leq 30 \\ S^{i+1}[31] &\leftarrow fb \end{aligned}$$

The boolean representation of keystream bit (obtained after applying Trace function on WGP-5) is given by $z_{i-64} = s^i_{155} + s^i_{156} + s^i_{157} + s^i_{158} + s^i_{159} + s^i_{155}s^i_{156} + s^i_{155}s^i_{157} + s^i_{155}s^i_{159} + s^i_{156}s^i_{158} + s^i_{156}s^i_{159} + s^i_{155}s^i_{156}s^i_{157} + s^i_{155}s^i_{157}s^i_{158} + s^i_{155}s^i_{157}s^i_{159} + s^i_{155}s^i_{158}s^i_{159} + s^i_{156}s^i_{157}s^i_{158} + s^i_{156}s^i_{158}s^i_{159}$. In what follows, we present the cube attack details on WG-5. The mathematical description and notations related to the cube attack are described in Section 2.4.4.2.

8.3 Cube Attack on WG-5

We adopt the techniques presented in [127] to analyze WG-5 with respect to cube attacks. The attack procedure is similar to [127] and consists of two phases: *offline phase* and *online phase*.

1. **Offline phase.** The goal of this phase is to recover a superpoly that is balanced for a given cube C_I. It consists of three steps:

   Step 1.1: Create a MILP model M for WG-5 whose initialization is reduced to R rounds. The model encodes the division property propagation for R rounds to check the feasibility of all R-round division trails.

   Step 1.2: Choose a cube C_I by flipping bits in $I = \{i_1, i_2, \ldots, i_{|I|}\}$ and then evaluate the secret variables involved in the superpoly. Let $J = \{k_{j_1}, k_{j_2}, \ldots, k_{j_{|J|}}\}$ denotes the set of involved secret variables[1].

   Step 1.3: Choose a value in the constant part of IV and compute $\bigoplus_{C_I} f(k, v) = p(\bar{k}, \bar{v})$, where $\bar{k} = \{k_{j_1}, k_{j_2}, \ldots, k_{j_{|J|}}\}$, $\bar{v} = \{v_0, v_1, \ldots, v_{79}\} \setminus \{v_{i_1}, v_{i_2}, \ldots, v_{i_{|I|}}\}$ and all the possible combinations of $k_{j_1}, k_{j_2}, \ldots, k_{j_{|J|}}$ are tried out, then $p(\bar{k}, \bar{v})$ is recovered and stored in a list for all values of $\bar{k}$. Assuming the best case (that we can recover the balanced superpoly in a single trial) the time complexity of this phase is bounded by $2^{|I|+|J|}$. However, if N cubes are used, the time complexity is given by $N2^{|I|+|J|}$.

2. **Online phase.** The goal of this phase is to recover the secret key. This phase is further divided into two steps.

   Step 2.1: Use the balanced superpoly recovered in the *offline phase* and query the cube C_I to the encryption oracle to obtain the value of $p(\bar{k}, \bar{v})$ which is then compared to the previously stored values. Then one bit is recovered from J (for example, if $p(\bar{k}, \bar{v}) = k_0 + k_1 + 1 = 0$, then $(k_0, k_1) = (1, 0)$ and $(k_0, k_1) = (0, 1)$ are the 2 possible key candidates out of 4 keys) as $p = 0$ for $2^{|J|-1}$ values and $p = 1$ for the remaining half values. To recover more than 1 bit we use multiple cubes.

   Step 2.2: Guess the remaining secret key bits.

8.3.1 Automating the cube attack on WG-5 using MILP

MILP model for the WGP-5 permutation. To model the WGP-5 permutation, we could use its boolean representation (Section 8.4). However, this approach results in large

[1]Step 1.2 is computationally feasible because of MILP

number of MILP variables and constraints due to its high nonlinearity and involvement of terms of up to degree 4 in each of the component functions. Hence, we use an alternative approach, we treat WGP-5 as a 5-bit Sbox. Let $(x_0, x_1, x_2, x_3, x_4)$ and $(y_0, y_1, y_2, y_3, y_4)$ be the input and output of the WGP-5 sbox, respectively. We use the *inequality_generator()* function in Sage [3] and Algorithms 1 and 2 in [134], and consequently find that only 12 inequalities are sufficient to model the division property propagation of WG-5. The inequalities are given by

$$\begin{cases} 2x_0 + 2x_1 + 2x_2 + 2x_3 + 6x_4 - 3y_0 - 3y_1 - 3y_2 - 3y_3 - 3y_4 \geq -1 \\ 4x_3 - y_0 - y_1 - y_2 - y_3 - y_4 \geq -1 \\ 4x_0 - y_0 - y_1 - y_2 - y_3 - y_4 \geq -1 \\ -x_0 - x_2 - x_3 - y_0 + 4y_1 - y_2 - y_3 - 2y_4 \geq -4 \\ -6x_0 - 3x_1 - 6x_3 - 6x_4 + 2y_0 - 4y_1 + 3y_2 - y_3 + 2y_4 \geq -19 \\ -3x_0 - x_1 - x_2 - 3x_3 - 2x_4 + 9y_0 + 7y_1 + 8y_2 + 9y_3 + 9y_4 \geq 0 \\ x_0 + x_1 + x_2 + x_3 + x_4 - 3y_0 - 3y_1 - 3y_2 - 3y_3 + 5y_4 \geq -2 \\ -x_0 - 3x_2 - 3x_3 - 2x_4 + y_0 + y_2 + y_3 - 2y_4 \geq -8 \\ -x_0 - x_1 + 2x_2 - x_3 - x_4 - y_0 - 2y_1 - 2y_2 + 3y_3 - y_4 \geq -5 \\ -x_0 - 2x_1 - 2x_2 - 2x_3 - x_4 - 2y_0 - y_1 - y_2 - y_3 + 5y_4 \geq -8 \\ -2x_0 - x_1 - 2x_2 - 2x_4 + y_0 + y_1 - y_2 + y_4 \geq -6 \\ -x_0 - x_2 - x_3 + y_0 - y_4 \geq -3. \end{cases}$$

Algorithm 8.1 describes the MILP model for the WGP-5.

Algorithm 8.1 MILP model for WGP-5

```
1: function WGP-5($S$)                                  ▷ $S = (s_0, s_1, \ldots, s_{159})$
2:     $M.var \leftarrow s'_{155+i}, x_i, y_i$ as binary for $0 \leq i \leq 4$
3:     $M.con \leftarrow s_{155+i} = s'_{155+i} + x_i$ for $0 \leq i \leq 4$
4:     Add constraints to $M$ according to the WGP-5 inequalities
5:     for $j = 0$ to 30 do
6:         $S'[j] = S[j]$
7:     end for
8:     return $(M, S', [y_0, y_1, y_2, y_3, y_4])$
9: end function
```

MILP model for the feedback function (FBK). The function FBK in Algorithm 8.2 generates the MILP variables and constraints for the feedback value $\gamma \otimes S^i[0] \oplus S^i[2] \oplus S^i[3] \oplus S^i[4] \oplus S^i[6] \oplus S^i[7]$. Since γ is constant, we model $\gamma S^i[0]$ as $S^i[0]$ for the sake of simplicity.

Algorithm 8.2 MILP model for the FBK function

```
1: function FBK($S, I$)
2:     for $i \in I$ do
3:         $M.var \leftarrow s'_{5i+j}, x_{5i+j}$ as binary for $0 \leq j \leq 4$
4:     end for
5:     $M.var \leftarrow y_i$ as binary for $0 \leq i \leq 4$
6:     for $i \in I$ do
7:         $M.con \leftarrow s_{5i+j} = s'_{5i+j} + x_{5i+j}$ for $0 \leq j \leq 4$
8:     end for
9:     for $j = 0$ to $4$ do
10:        $temp = 0$
11:        for $i \in I$ do
12:            $temp = temp + x_{5i+j}$
13:        end for
14:        $M.con \leftarrow y_j = temp$
15:    end for
16:    for $j \in \{(0, 1, \ldots, 31) - I\}$ do
17:        $S'[j] = S[j]$
18:    end for
19:    return $(M, S', [y_0, y_1, y_2, y_3, y_4])$
20: end function
```

MILP model for KSG. The function KSG in Algorithm 8.3 creates the MILP variables and constraints for the keystream bit $z = s^R_{155} + s^R_{156} + s^R_{157} + s^R_{158} + s^R_{159} + s^R_{155}s^R_{156} + s^R_{155}s^R_{157} + s^R_{155}s^R_{159} + s^R_{156}s^R_{158} + s^R_{156}s^R_{159} + s^R_{155}s^R_{156}s^R_{157} + s^R_{155}s^R_{157}s^R_{158} + s^R_{155}s^R_{157}s^R_{159} + s^R_{155}s^R_{158}s^R_{159} + s^R_{156}s^R_{157}s^R_{158} + s^R_{156}s^R_{158}s^R_{159}$. Furthermore, the bitwise AND and XOR operations are modeled using Algorithm 8.4.

Algorithm 8.3 MILP model for the KSG

```
1: function KSG($S$)
2:     $(M, S_1, a_1) = \text{AND}(S, [155, 156])$
3:     $(M, S_2, a_2) = \text{AND}(S_1, [155, 157])$
4:     $(M, S_3, a_3) = \text{AND}(S_2, [155, 159])$
5:     $(M, S_4, a_4) = \text{AND}(S_3, [156, 158])$
6:     $(M, S_5, a_5) = \text{AND}(S_4, [156, 159])$
7:     $(M, S_6, a_6) = \text{AND}(S_5, [155, 156, 157])$
8:     $(M, S_7, a_7) = \text{AND}(S_6, [155, 157, 158])$
9:     $(M, S_8, a_8) = \text{AND}(S_7, [155, 157, 159])$
10:    $(M, S_9, a_9) = \text{AND}(S_8, [155, 158, 159])$
11:    $(M, S_{10}, a_{10}) = \text{AND}(S_9, [156, 157, 158])$
12:    $(M, S_{11}, a_{11}) = \text{AND}(S_{10}, [156, 158, 159])$
13:    $(M, S_{12}, a_{12}) = \text{XOR}(S_{11}, [155, 156, 157, 158, 159])$
14:    $M.var \leftarrow z$ as binary
15:    $M.con \leftarrow z = \sum_{i=1}^{12} a_i$
16:    return $(M, S_{12}, z)$
17: end function
```

MILP model for WG-5. The MILP model for WG-5 is given in Algorithm 8.5. It incorporates the previous models for WGP-5, FBK and KSG. The function WG5eval in Algorithm 8.5 evaluates all division trails for WG-5 whose initialization round is reduced to R. The number of MILP variables and constraints required for each function are given in Table 8.1.

Table 8.1: MILP variables and constraints

Function	# of variables	# of constraints
WGP-5	15	17
FBK	65	35
KSG	79	63
R round of WG-5	$160 + 159R + 5R$	$161 + 115R + 10R$

8.3.2 Evaluating involved secret variables and superpoly recovery

We prepare a cube C_I by flipping bits in $I = \{i_1, i_2, \ldots, i_{|I|}\}$, and then evaluate the involved secret variables in superpoly using the generic algorithm proposed in [127]. We have given the description of the utilized algorithm (Algorithm 8.6) for the sake of completeness. The inputs to Algorithm 8.6 are the cube indices set I and the MILP model M for WG-5. The

Algorithm 8.4 MILP model for AND and XOR operations

```
1: function AND($S, I$)
2:     $M.var \leftarrow s'_i, x_i$ as binary for $i$ in $I$
3:     $M.var \leftarrow y$ as binary
4:     $M.con \leftarrow s_i = s'_i + x_i$ for $i$ in $I$
5:     $M.con \leftarrow y \geq x_i$ for $i$ in $I$
6:     for $i \in \{(0, 1, \ldots, 159) - I\}$ do
7:         $s'_i = s_i$
8:     end for
9:     return $(M, S', y)$
10: end function
11: function XOR($S, I$)
12:     $M.var \leftarrow s'_i, x_i$ as binary for $i$ in $I$
13:     $M.var \leftarrow y$ as binary
14:     $M.con \leftarrow s_i = s'_i + x_i$ for $i$ in $I$
15:     $temp = 0$
16:     for $i \in I$ do
17:         $temp = temp + x_i$
18:     end for
19:     $M.con \leftarrow y = temp$
20:     for $i$ in $\{(0, 1, \ldots, 159) - I\}$ do
21:         $s'_i = s_i$
22:     end for
23:     return $(M, S', y)$
24: end function
```

model M evaluates all the division trails for R rounds with input division property given by $v_i = 1$ for $i \in I$ and $v_i = 0$ for $i \in \{\{0, 1, \ldots, 79\} \setminus I\}$.

Algorithm 8.5 MILP model for the initialization phase of WG-5

```
1: function WG5eval($R$)
2:     Prepare empty MILP Model $M$
3:     $M.var \leftarrow S^0[j]$ for $0 \leq j \leq 31$
4:     for $i = 1$ to $R$ do
5:         $(M, S', a) =$ WGP-5$(S^{i-1})$
6:         $(M, S'', b) =$ FBK$(S', [0, 2, 3, 4, 6, 7])$
7:         for $j = 0$ to $30$ do
8:             $S^i[j] = S''[j + 1]$
9:         end for
10:        $M.con \leftarrow S''[0] = 0$
11:        $M.var \leftarrow S^i[31]$ as binary
12:        $M.con \leftarrow S^i[31] = a + b$
13:    end for
14:    $(M, S''', z) =$ KSG$(S^R)$
15:    for $j = 0$ to $31$ do
16:        $S'''[j] = 0$
17:    end for
18:    $M.con \leftarrow z = 1$
19: end function
```

Algorithm 8.6 MILP model to find involved secret variables in superpoly [127]

```
1: function extractSecretVariables(MILP model $M$, Cube Indices $I$)
2:     $M.var \leftarrow k_i$ as binary for $0 \leq i \leq n - 1$,      ▷ $k_0, k_1, \ldots, k_{n-1}$ are secret variables
3:     $M.var \leftarrow v_i$ as binary for $0 \leq i \leq m - 1$,      ▷ $v_0, v_1, \ldots, v_{m-1}$ are public variables
4:     $M.con \leftarrow v_i = 1$ for $i \in I$
5:     $M.con \leftarrow v_i = 0$ for $i \in \{(0, 1, \ldots, m - 1) - I\}$
6:     $M.con \leftarrow \sum_{i=0}^{n-1} k_i = 1$
7:     do
8:         solve MILP model $M$
9:         if $M$ is feasible then
10:            pick $j \in \{0, 1, \ldots, n - 1\}$ s.t $k_j = 1$
11:            $J = J \cup \{j\}$
12:            $M.con \leftarrow k_j = 0$
13:        end if
14:    while $M$ is feasible
15:    return $J$
16: end function
```

Table 8.2: Involved secret variables in superpoly for cube indices $I \in \{I_1, I_2, I_3, I_4, I_5\}$

Rounds	Involved secret variables J	Time complexity $\log_2(\cdot)$
15	$\{k_5, k_6, \ldots, k_{54}\}$	54
16	$\{k_5, k_6, \ldots, k_{54}\}$	54
17	$\{k_5, k_6, \ldots, k_{59}\}$	59
18	$\{k_5, k_6, \ldots, k_{59}\}$	59
19	$\{k_5, k_6, \ldots, k_{64}\}$	64
20	$\{k_5, k_6, \ldots, k_{64}\}$	64
21	$\{k_5, k_6, \ldots, k_{69}\}$	69
22	$\{k_5, k_6, \ldots, k_{69}\}$	69
23	$\{k_5, k_6, \ldots, k_{74}\}$	74
24	$\{k_5, k_6, \ldots, k_{74}\}$	74

$I_1 = \{0, 1, 2, 3\}, I_2 = \{0, 1, 2, 4\}, I_3 = \{0, 1, 3, 4\}, I_4 = \{0, 2, 3, 4\}, I_5 = \{1, 2, 3, 4\}$

Here, time complexity means the complexity to recover the superpoly.

Searching cubes. We limit our search of the cubes to indices I such that $2^{|I|+|J|} < 2^{80}$. Table 8.2 lists the cubes we found that satisfy this condition. Note that searching all $\binom{80}{|I|}$ cubes is infeasible and the cubes in Table 8.2 are the best so far for WG-5 according to our experimental results.

Recovering a balanced superpoly. We choose a value in the constant part of the IV and vary all $2^4 \times 2^{70}$ values to recover $p(k_5, k_6, \ldots, k_{74}, \bar{v})$ where $\bar{v} = \{v_0, v_1, \ldots, v_{79}\} \setminus \{v_j |\ j \in I_i\}$ for $1 \leq i \leq 5$ and $R = 24$. We also store 2^{70} values of $p(k_5, k_6, \ldots, k_{74}, \bar{v})$ as they will be used again in the online phase. We assume that we can recover a balanced superpoly in 1 trial for each of the cubes in Table 8.2. If not, we repeat the experiment.

8.3.3 Theoretical key recovery attack on 24 rounds

We use the balanced superpolys recovered in the offline phase for cubes I_1, I_2, I_3, I_4 and I_5 (Table 8.2) in the online phase. We query the cube C_{I_i} to the encryption oracle and compute the sum $\bigoplus_{C_{I_i}} f(k, v)$. We then compare this sum with $\bigoplus_{C_{I_i}} f(k, v) = p(k_5, k_6, \ldots, k_{74}, \bar{v})$ stored in the offline phase for all possible combinations of $\{k_5, k_6, \ldots, k_{74}\}$. We discard the values of $\{k_5, k_6, \ldots, k_{74}\}$ for which the sum is different. Since we are using a balanced superpoly, $p(k_5, k_6, \ldots, k_{74}, \bar{v}) = 0$ for 2^{69} values and equals 1 for the remaining 2^{69} values. Thus, one bit of secret information can always be recovered. We use cubes I_1, I_2, I_3, I_4 and I_5 in our attack and hence can recover 5 secret variables. We then guess the remaining 75 bits to recover the entire secret key. The attack time complexity for 24 rounds equals $5 \times 2^{74} + 2^{75} \approx 2^{76.81}$.

8.3.4 Attack comparison with algebraic attacks

The univariate algebraic attacks [118] exploit the fact that WG-5 is updated linearly during the keystream generation phase (Section 8.2.2). Hence, using the trace representation of z_t, it is possible to find a low degree multiple g (also known as annihilator) of filtering function f, i.e., $fg = 0$ which lowers the data and time complexity of the algebraic attack to 2^{15} and 2^{33}, respectively. Such attacks do not hold if the nonlinear WGP-5 is fed back into the state during KSG phase because the idea of annihilators no longer exists. On the other hand, our attack is not affected by this fact and it only requires significantly low data complexity.

8.4 Comparison of the Initialization Phases

In this section, we present an argument to show how the initialization phase of WG-5 is more resistant to cube attacks than those of Grain-128a [11] and Trivium[53].

8.4.1 Brief description of Grain128 and Trivium

Grain-128a is an NLFSR based stream cipher of Grain family with two 128-bit states represented by $(b_0, b_1, \ldots, b_{127})$ and $(s_0, s_1, \ldots, s_{127})$. The state is loaded with a 128-bit key and a 96-bit IV as follows.

$$\begin{aligned}
&(b_0, b_1, \ldots, b_{127}) \leftarrow (k_0, k_1, \ldots, k_{127}),\\
&(s_0, s_1, \ldots, s_{127}) \leftarrow (iv_0, iv_1, \ldots, iv_{95}, 1, \ldots, 1, 0).
\end{aligned}$$

The initialization phase runs for 256 rounds with the state update function given by

$$\begin{aligned}
g \leftarrow\ & b_0 + b_{26} + b_{56} + b_{91} + b_{96} + b_3 b_{67} + b_{11} b_{13}\\
&+ b_{17} b_{18} + b_{27} b_{59} + b_{40} b_{48} + b_{61} b_{65} + b_{68} b_{84}\\
&+ b_{88} b_{92} b_{93} b_{95} + b_{22} b_{24} b_{25} + b_{70} b_{78} b_{82}\\
f \leftarrow\ & s_0 + s_7 + s_{38} + s_{70} + s_{81} + s_{96}\\
h \leftarrow\ & b_{12} s_8 + s_{13} s_{20} + b_{95} s_{42} + s_{60} s_{79} + b_{12} b_{95} s_{94}\\
z \leftarrow\ & h + s_{93} + b_2 + b_{15} + b_{36} + b_{45} + b_{64} + b_{73} + b_{89}\\
(b_0, b_1, \ldots, b_{127}) \leftarrow\ & (b_1, b_2, \ldots, b_{127}, g + s_0 + z)\\
(s_0, s_1, \ldots, s_{127}) \leftarrow\ & (s_1, s_2, \ldots, s_{127}, f + z).
\end{aligned}$$

During the KSG phase, z is not fed back to the state and is directly used as the keystream bit.

Trivium is another NLFSR based stream cipher with state size 288. The 80-bit key and 80-bit IV are loaded into the state as follows: $(s_0, s_1, \ldots, s_{92}) \leftarrow (k_0, k_1, \ldots, k_{79}, 0, \ldots, 0)$,

$(s_{93}, s_{94}, \ldots, s_{176}) \leftarrow (iv_0, iv_1, \ldots, iv_{79}, 0, \ldots, 0)$ and $(s_{177}, s_{178}, \ldots, s_{287}) \leftarrow (0, 0, \ldots, 0, 1, 1, 1)$. The state update function of Trivium is given by

$$\begin{aligned}
&t_1 \leftarrow s_{65} + s_{92}\\
&t_2 \leftarrow s_{161} + s_{176}\\
&t_3 \leftarrow s_{242} + s_{287}\\
&z \leftarrow t_1 + t_2 + t_3\\
&t_1 \leftarrow t_1 + s_{90}s_{91} + s_{170}\\
&t_2 \leftarrow t_2 + s_{174}s_{175} + s_{263}\\
&t_3 \leftarrow t_3 + s_{285}s_{286} + s_{68}\\
&(s_0, s_1, \ldots, s_{92}) \leftarrow (t_3, s_0, \ldots, s_{91})\\
&(s_{93}, s_1, \ldots, s_{176}) \leftarrow (t_1, s_{93}, \ldots, s_{175})\\
&(s_{177}, s_1, \ldots, s_{287}) \leftarrow (t_2, s_{177}, \ldots, s_{286}).
\end{aligned}$$

The initialization phase runs for 1152 rounds without producing an output while z is used as the keystream bit during KSG phase.

8.4.2 Observations on degree evaluation

Trivium. The degree of z is 3 after 81 rounds. The algebraic degree of z can only be increased by AND terms $s_{90}s_{91}$, $s_{174}s_{175}$ and $s_{285}s_{286}$. Thus, the round at which the degree of z equals 3 is $\min(90, 174 - 93, 285 - 177) = 81$.

Grain128a. The degree of z is 6 after 32 rounds. The maximum index in h function is 95 (for b_{95} term). At round 32 (127-95) only the degree of b_{95} is 4 and the remaining terms are of degree 1. Hence, the degree of z is 6 because of the $b_{12}b_{95}s_{94}$ term.

WG-5. The degree of z is 6 in 1 round. Note that the degree of each component of $S^1[31]$ equals 4. This can be deduced from the boolean representation of the component functions of the WGP-5 which is given below.

$$\begin{aligned}
y_0 = {} & x_0x_1x_3x_4 + x_0x_1x_4 + x_0x_2x_3x_4 + x_0x_2x_3 + x_0x_2x_4\\
& + x_0x_4 + x_0 + x_1x_2x_3 + x_1x_2 + x_1x_3 + x_3x_4\\
y_1 = {} & x_0x_1x_2x_3 + x_0x_1x_2x_4 + x_0x_1x_2 + x_0x_1x_3 + x_0x_1x_4 + x_0x_1 + x_0x_2x_4\\
& + x_0x_2 + x_0x_3x_4 + x_0x_4 + x_1x_2x_3x_4 + x_1x_4 + x_1 + x_2x_4 + x_2 + x_3x_4
\end{aligned}$$

$$\begin{aligned} y_2 = {} & x_0x_1x_2x_3 + x_0x_1x_4 + x_0x_1 + x_0x_2 + x_0x_3x_4 + x_1x_2x_3x_4 + x_1x_2 + x_1x_4 \\ & + x_2x_3x_4 + x_2x_3 + x_2x_4 + x_2 + x_3x_4 + x_3 + x_4 \\ y_3 = {} & x_0x_1x_2x_3 + x_0x_1x_3 + x_0x_1 + x_0x_2x_3x_4 + x_0x_2x_3 + x_0x_2x_4 \\ & + x_0x_3x_4 + x_0x_4 + x_1x_2x_4 + x_1x_3x_4 + x_1x_3 + x_1 \\ y_4 = {} & x_0x_1x_2x_4 + x_0x_1x_2 + x_0x_1x_3x_4 + x_0x_1 + x_0x_2x_3x_4 + x_0x_2 + x_0x_3x_4 \\ & + x_0x_3 + x_0x_4 + x_1x_2x_3 + x_1x_2x_4 + x_1x_2 + x_1x_3 + x_1x_4 \\ & + x_1 + x_2x_3x_4 + x_2x_3 + x_2x_4 + x_4 \end{aligned}$$

Since z at round 1 is given by $s^1_{155} + s^1_{156} + s^1_{157} + s^1_{158} + s^1_{159} + s^1_{155}s^1_{156} + s^i_{155}s^1_{157} + s^1_{155}s^1_{159} + s^1_{156}s^i_{158} + s^i_{156}s^i_{159} + s^1_{155}s^1_{156}s^1_{157} + s^1_{155}s^1_{157}s^1_{158} + s^1_{155}s^1_{157}s^1_{159} + s^1_{155}s^1_{158}s^1_{159} + s^1_{156}s^1_{157}s^1_{158} + s^1_{156}s^1_{158}s^1_{159}$, then the degree of z is $4 + 2 = 6$.

Based on the degree comparison of 32 rounds of Grain-128a and 81 rounds of Trivium with 1 round of WG-5, the degree in WG-5 grows faster because of the state update by the nonlinearly generated bits at each clock cycle, which is not the case with Grain-128a and Trivium. We also observe that all the 5 bits processed by WGP-5 at the i-th round are used to generate the keystream bit at round $(i+1)$ along with $5 \times 6 = 30$ new bits from the feedback function. This is not the same case with Grain-128 because the updated bits b_{127} and s_{127} in i-th round are used in keystream bit at $i + 32$ and $i + 33$, respectively. Similarly, for Trivium the values of t_1, t_2 and t_3 at i-th round are used in keystream bit at $i + 90, i + 81$ and $i + 108$ rounds, respectively. Thus, we expect that initialization phase of WG-5 is stronger than those of Grain-128a and Trivium with respect to cube attacks.

8.5 Summary

In this chapter, we have investigated the security of initialization phase of WG-5 with respect to cube attacks. We have modeled the division trails of reduced-round WG-5 using MILP and have shown a key recovery attack on 24 rounds with data and time complexity of $2^{6.32}$ and $2^{76.81}$, respectively. Finally, we have provided an argument to show that the WG-5 design parameters in terms of feedback and tap positions are more resistant to cube attacks in contrast to Grain-128a and Trivium.

Chapter 9

Meet-in-the-Middle Attack using Correlated Sequences and its Applications to Simon-like Ciphers

Contents

9.1 Motivation

The meet-in-the-middle (MitM) attack is a well-known generic cryptanalytic technique against cryptographic primitives. It has broadly two types: the standard Diffie-Hellman MitM attack [57] and the multi-dimensional MitM attack proposed by Zhu and Gong [141]. Both variants are highly dependent on the matching phase. More precisely, for a matching phase, the key space has to be partitioned into l independent sets if the cipher is decomposed as a composition of l subciphers (Section 2.4.5, Figure 2.26 with $l = 4$). This clearly has a limitation as the matching variable/variables can only be found for a small number of rounds, unless the key scheduling algorithm is weak. However, most of the key scheduling algorithms are designed in a way so that key bits are mixed properly in few rounds. Accordingly, the number of rounds covered by MitM attacks is usually lower than that of differential and linear, and algebraic attacks.

In this chapter, we introduce a novel characteristic of block ciphers called correlated sequences. We apply the method of correlated sequences to propose a generic MitM attack on Feistel and SPN ciphers which could cover more rounds than the traditional MitM, differential and linear, and algebraic attacks. As an application of our attack, we break up to 85% (27/32) of the rounds of two lightweight block ciphers Simon-32/64 and Simeck-32/64 (Section 2.5.3) while the previous attacks can reach only 23 rounds (Table 9.1). Although, our attack on these ciphers has time complexity close to that of integral attacks which is 2^{63} encryptions, it has a very low data complexity and success probability 1. In addition, it depicts weaknesses which were not investigated before.

Table 9.1: State-of-the-art attacks on Simon-32/64 and Simeck-32/64

Attack	**Cipher**	**# attacked rounds**	**Data**	**Memory**	**Time**	**Success rate**
		out of 32		(bytes)		
Differential	Simon-32/64 [130]	21	2^{31}	-	$2^{55.25}$	0.51
	Simon-32/64 [109]	22	2^{32}	-	$2^{58.76}$	0.315
	Simeck-32/64 [87]	19	2^{31}	2^{33}	2^{40}	-
	Simeck-32/64 [109]	22	2^{32}	-	$2^{57.9}$	0.417
Linear	Simon-32/64 [47]	23	$2^{31.19}$	-	$2^{61.84}$	0.277
	Simeck-32/64 [110]	23	$2^{31.91}$	-	$2^{61.78}$	0.456
Integral	Simon-32/64 [131]	21	2^{31}	2^{54}	2^{63}	1
	Simon-32/64 [66]	22	2^{31}	$2^{55.8}$	2^{63}	1
	Simon-32/64 [49]	24	2^{32}	$2^{33.64}$	2^{63}	1
	Simeck-32/64 [138]	21	2^{31}	$2^{46.22}$	2^{63}	1
Impossible Differential	Simon-32/64 [54]	20	2^{32}	$2^{45.5}$	$2^{62.8}$	-
	Simeck-32/64 [135]	20	2^{32}	2^{58}	$2^{62.5}$	-
Zero correlation	Simon-32/64 [123]	21	2^{32}	2^{31}	$2^{59.4}$	-
	Simeck-32/64 [139]	21	2^{32}	$2^{47.67}$	$2^{58.78}$	-
Meet-in-the-middle	Simon-32/64 [120]	18	8	2^{52}	$2^{62.57}$	1
Correlated sequences Sections 9.4 and 9.5	Simon-32/64	**24**	3	2^{49}	$2^{62.87}$	1
		25	3	2^{49}	$2^{62.94}$	1
		26	3	2^{49}	$2^{62.88}$	1
		27	3	2^{49}	$2^{62.94}$	1
	Simeck-32/64	**24**	3	2^{49}	$2^{62.87}$	1
		25	3	2^{49}	$2^{62.94}$	1
		26	3	2^{49}	$2^{62.88}$	1
		27	3	2^{49}	$2^{62.94}$	1

Outline. The rest of the chapter is organized as follows. Section 9.2 details the definitions and basic properties of the correlated sequences, and applications of these sequences to MitM attacks on Feistel and SPN ciphers. In Section 9.3, we provide the theoretical construction of correlated sequences for Simon-like ciphers. Section 9.4 presents the application of correlated sequences to a 25-round key recovery attack on Simon-32/64 and Simeck-32/64. In Section 9.5, we extend the 25-round attack to 27 rounds using the key scheduling algorithm properties.

9.2 Correlated Sequences of Block Ciphers

In this section, we formally introduce the correlated sequences and then show how to use them in the meet-in-the-middle attack (Section 2.4.5). Consider an n-bit block cipher with r rounds and an mn-bit master key $k = (k_0, k_1, \ldots, k_{m-1})$ as depicted in Figure 9.1. Let s_i denote the state at the i-th round. Then for $0 \leq i < r$, $s_{i+1} = \mathcal{RF}(s_i, k_i)$ where $\mathcal{RF}$ denotes the round function (Section 2.3.2.2) and it is generally a composition of two functions, namely 1) a linear function $\mathcal{L}$ and 2) a nonlinear function $\mathcal{N}$. Note that the order of composition, i.e., $\mathcal{L} \circ \mathcal{N}$ or $\mathcal{N} \circ \mathcal{L}$ is cipher dependent and we omit it for the general description.

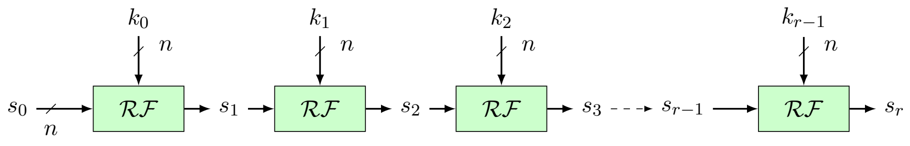


Figure 9.1: A generic diagram of a block cipher

9.2.1 General idea

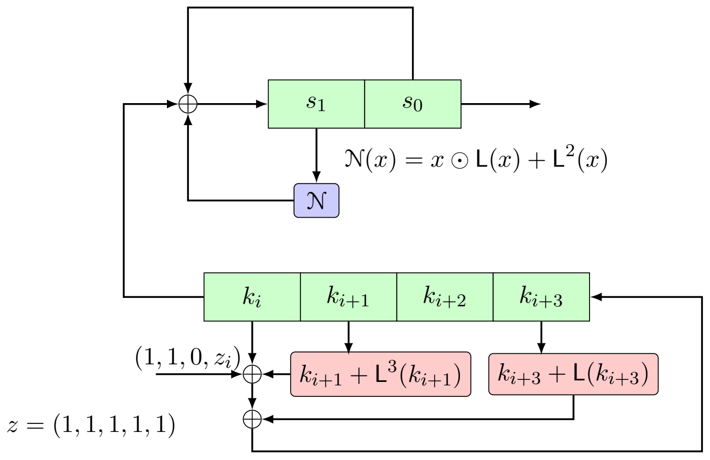
Definition 9.1 (Keyed sequence). Given $k \in \mathcal{K}$ and $1 \leq t < r$, we say that $S_{(k,t)} = (s_0, s_1, \ldots, s_{t-1})$ is a *keyed sequence* of length t if $s_{i+1} = \mathcal{RF}(s_i, k_i)$ for $0 \leq i < t-1$.

From Definition 9.1, it is clear that we need to compute $\mathcal{RF}$ t times to obtain $S_{(k,t)}$. This implies that $\mathcal{N}$ is computed t times in total. Thus, to obtain another sequence $S_{(k',t)}$ of the same length t, the worst case is to compute $\mathcal{N}$ exactly t times. The idea of correlated sequences is **"Given $S_{(k,t)}$ and $k' \neq k$, obtain the sequence $S_{(k',t)}$ by computing the nonlinear function $\mathcal{N}$ less than t times."** We present a toy example to illustrate this notion before providing the formal definition. In the following, we use "+" to denote bitwise XOR and integer addition if the meaning is clear in the context.

Example 9.1. Consider a 4-bit toy Simon-like block cipher with 8-bit blocksize and 16-bit key as shown in Figure 9.2. Let the nonlinear function be given by $\mathcal{N}(x) = x \odot \mathsf{L}(x) + \mathsf{L}^2(x)$ where $x \in \mathbb{F}_2^4$ and $\mathsf{L}^i(\cdot)$ denotes the left cyclic shift by i bits. The length seven keyed sequences when

$k_0 = 0$ and $k_1 = 0$ are given in Table 9.2 (values given in integers). We note the following observations from Table 9.2.

1. For all $k = (k_0, k_1, k_2, k_3)$, $s_4 = k_2$, $s_5 = 0$ and $s_6 = k_2 + k_4$.
2. For all $k' = (k'_0, k'_1, k'_2.k'_3)$, $s'_4 = k'_2$, $s'_5 = 1$ and $s'_6 = k'_2 + k'_4 + \mathcal{N}(1)$.
3. For each row, $k'_3 = k_3 + 1$ and $s'_6 = s_6 + k_4 + k'_4 + \mathcal{N}(1)$.



s_1
s_0
$\mathcal{N}(x) = x \odot \mathsf{L}(x) + \mathsf{L}^2(x)$
$\mathcal{N}$
k_i
k_{i+1}
k_{i+2}
k_{i+3}
$(1, 1, 0, z_i)$
$k_{i+1} + \mathsf{L}^3(k_{i+1})$
$k_{i+3} + \mathsf{L}(k_{i+3})$
$z = (1, 1, 1, 1, 1)$

Figure 9.2: 4-bit toy Simon-like cipher

We now define the *correlated sequences* in Definition 9.2.

Definition 9.2 ((σ, t)-correlated sequences)**.** Given $S_{(k,t)}$ and $0 \le \sigma < t$, we say $S_{(k,t)}$ and $S_{(k',t)}$ are (σ, t)-correlated sequences if $S_{(k',t)}$ can be obtained from $S_{(k,t)}$ by computing the nonlinear function $\mathcal{N}$ exactly σ times.

Remark 9.1. $\sigma = 0 \implies S_{(k,t)}$ and $S_{(k',t)}$ are linearly related.

Definition 9.3 (Linear correlated keys)**.** Given $S_{(k,t)}$, we define *linear correlated keys* as the set

$$\mathrm{CK}(k) = \{k' \mid S_{(k,t)} \text{ and } S_{(k',t)} \text{ are } (0,t)\text{-correlated sequences}\}.$$

For example, in Table 9.2, for each row $S_{(k,t)}$ and $S_{(k',t)}$ are (1,7) correlated sequences. Further $|\mathrm{CK}((0,0,0,0))| = 15$ and $|\mathrm{CK}((0,0,0,1))| = 15$, i.e., the last 15 sequences can be computed linearly from the sequence corresponding to the keys $(0, 0, 0, 0)$ and $(0, 0, 0, 1)$, respectively. More precisely, to obtain all 32 sequences, we only need to compute $\mathcal{N}$ exactly once.

Table 9.2: Keyed sequences (values given in integers)

k_0	k_1	k_2	k_3	k_4	s_0	s_1	s_2	s_3	s_4	s_5	s_6	k'_0	k'_1	k'_2	k'_3	k'_4	s'_0	s'_1	s'_2	s'_3	s'_4	s'_5	s'_6
0	**0**	**0**	**0**	**13**	**0**	**0**	**0**	**0**	**0**	**0**	**13**	**0**	**0**	**0**	**1**	**14**	**0**	**0**	**0**	**0**	**0**	**1**	**10**
0	0	1	4	1	0	0	0	0	1	0	0	0	0	1	5	2	0	0	0	0	1	1	7
0	0	2	8	4	0	0	0	0	2	0	6	0	0	2	9	7	0	0	0	0	2	1	1
0	0	3	14	14	0	0	0	0	3	0	13	0	0	3	15	13	0	0	0	0	3	1	10
0	0	4	1	14	0	0	0	0	4	0	10	0	0	4	0	13	0	0	0	0	4	1	13
0	0	5	5	2	0	0	0	0	5	0	7	0	0	5	4	1	0	0	0	0	5	1	0
0	0	6	13	11	0	0	0	0	6	0	13	0	0	6	12	8	0	0	0	0	6	1	10
0	0	7	11	1	0	0	0	0	7	0	6	0	0	7	10	2	0	0	0	0	7	1	1
0	0	8	2	11	0	0	0	0	8	0	3	0	0	8	3	8	0	0	0	0	8	1	4
0	0	9	7	4	0	0	0	0	9	0	13	0	0	9	6	7	0	0	0	0	9	1	10
0	0	10	10	2	0	0	0	0	10	0	8	0	0	10	11	1	0	0	0	0	10	1	15
0	0	11	13	11	0	0	0	0	11	0	0	0	0	11	12	8	0	0	0	0	11	1	7
0	0	12	11	1	0	0	0	0	12	0	13	0	0	12	10	2	0	0	0	0	12	1	10
0	0	13	14	14	0	0	0	0	13	0	3	0	0	13	15	13	0	0	0	0	13	1	4
0	0	14	7	4	0	0	0	0	14	0	10	0	0	14	6	7	0	0	0	0	14	1	13
0	0	15	0	13	0	0	0	0	15	0	2	0	0	15	1	14	0	0	0	0	15	1	5

9.2.2 Applications to MitM attack

Let (s_0, s_r) denote the plaintext and ciphertext pair encrypted with the mn-bit master key. As depicted in Figure 9.3, we first use s_0 to construct (σ, t)-correlated sequences and their corresponding CK$(\cdot)$ for t rounds. Next, starting with s_r, we follow the same approach. We then do partial encryption for l rounds starting from t-th round and match the state values at $(t+l)$-th round. Thus, when $\mathcal{RF}$ is an SPN round, the number of attacked rounds is $2t+l$. For a Feistel $\mathcal{RF}$ the number of attacked rounds is $2t-4+l$ (length t sequence corresponds to the output of $(t-2)$-th round), and equals $2t-3+l$ if matching is done on half state.

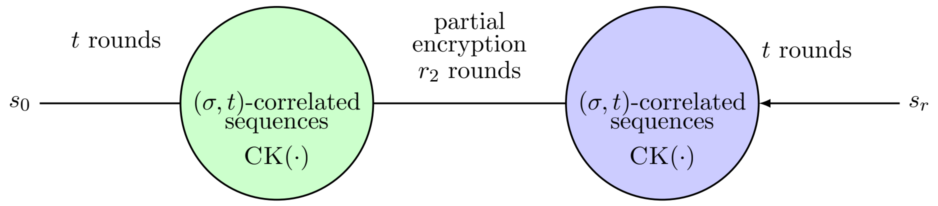

t rounds
s_0
(σ, t)-correlated sequences
CK$(\cdot)$
partial encryption r_2 rounds
(σ, t)-correlated sequences
CK$(\cdot)$
t rounds
s_r

Figure 9.3: MitM attack using correlated sequences

We summarize the above discussion in the following proposition.

Proposition 9.1. If there are (σ, t)-correlated sequences, then there exists an MitM attack for $2t - 3 + l$ (resp. $2t + l$) rounds for Feistel (resp. SPN) ciphers, where l is the number of rounds of partial encryption.

Time complexity. Let T^e (resp. T^d) denote the number of computations of $\mathcal{N}$ to construct (σ, t)-correlated sequences and their corresponding $\mathrm{CK}(\cdot)$ in encryption (resp. decryption) direction. Then, the time complexity in terms of the number of computations of $\mathcal{N}$ is given by $T = T^e + T^d + |\mathcal{K}| \times \frac{l}{r}$ where $\mathcal{K}$ denotes the space of keys. Clearly, $T < |\mathcal{K}|$ if $T^e + T^d \ll |\mathcal{K}|$.

Data complexity. The above attack filters $2^{n(m-1)}$ keys that map s_0 to s_r. The correct key can then be found out by performing an exhaustive search on the remaining known $m-1$ plaintext-ciphertext pairs. Note that for a Feistel cipher, an additional plaintext-ciphertext pair is needed if matching is done on half state.

9.3 Correlated Sequences of Simon-like Ciphers

In this section, we show the construction of correlated sequences of Simon-like ciphers (Section 2.5.3) where the key length is twice the block size. We first look at the theoretical properties of nonlinear function $f_{(a,b,c)}$. Next, we use these properties to construct $(1, 8)$-correlated sequences. We assume that $a \neq b \neq c$.

9.3.1 Properties of simon-like nonlinear function

Property 9.1. Let s be the coset leader corresponding to the coset C_s. Then for $0 \leq i < |C_s|$, the following assertions hold.

1. $f_{(a,b,c)}(\mathsf{L}^i(s)) = \mathsf{L}^i(f_{(a,b,c)}(s))$
2. $f_{(a,b,c)}(s) = \mathsf{L}^{a-1}(s) + \mathsf{L}^{b-1}(s) + \mathsf{L}^{c-1}(s)$ if $s = \underbrace{011\ldots1}_{n}$.

Property 9.2. Let $s = \underbrace{0101\ldots01}_{n}$ and a, b are not both simultaneously even or odd. Then

$$f_{(a,b,c)}(s) = \begin{cases} s & \text{if } c \equiv 0 \bmod 2 \\ \mathsf{L}(s) & \text{otherwise.} \end{cases}$$

Properties 9.1 and 9.2 imply that it is enough to compute the values of $f_{(a,b,c)}$ for coset leaders[1] only. As $f_{(a,b,c)}$ is quadratic and the only linear term involved in it is $\mathsf{L}^c(.)$, we have $f_{(a,b,c)}(x) = \mathsf{L}^c(x) + z$ for all $x \in \mathbb{F}_2^n$ and some constant $z \in \mathbb{F}_2^n$. As a result, we partition the coset leaders based on the values of z. Since $f_{(a,b,c)}$ is linear on each partition, we call such a partition a *z-linear segment set* and formally define it in Definition 9.4 as follows.

[1]Number of coset leaders $\approx \frac{2^n-1}{n}$

Definition 9.4 (z-linear segment set)**.** A z-linear segment set of $f_{(a,b,c)}$ is the set of coset leaders CL_z given by

$$CL_z = \{s \mid f_{(a,b,c)}(s) + \mathsf{L}^c(s) = z\}.$$

Table 9.3 lists the z-linear segment sets for $n = 8$ and $(a, b, c) = (8, 1, 2)$ while the number of z-linear segments (denoted by N_z) for varying n are presented in Table 9.4. (Note that since $n = 8$, the shifts $(8, 1, 2)$ is equivalent to $(0, 1, 2)$.)

Table 9.3: z-linear segment sets for $n = 8$ and $(a, b, c) = (8, 1, 2)$

z	CL_z	z	CL_z
0	{0, 1, 5, 9, 17, 21, 37, 85}	2	{3, 11, 19, 43}
6	{7, 23, 39, 87 }	8	{13, 45}
14	{15, 47}	16	{25}
18	{27, 91 }	24	{29}
30	{31, 95}	32	{53}
34	{51}	38	{55}
50	{59}	56	{61}
62	{63}	78	{111}
102	{119}	126	{127}
255	{255}	-	-

Table 9.4: Number of z-linear segment sets for varying n

n	# coset leaders	N_z	
		(a, b, c)	
		$(8, 1, 2)$	$(5, 0, 1)$
8	36	20	17
10	108	42	14
12	352	119	119
14	1182	50	287
16	4116	909	798

Example 9.2. In Table 9.3, consider $z = 2$ and $3 \in CL_2$. Then for all the elements of coset with coset leader 3, i.e.,

$$x \in C_3 = \{3, 6, 12, 24, 48, 96, 192, 129\},$$

the computation of $f_{(8,1,2)}(\cdot)$ is listed in Table 9.5.

Table 9.5: Computation of $f_{(8,1,2)}(x)$ for $x \in C_3$

x	$f_{(8,1,2)}(x)$
3	$\mathsf{L}^2(3) + 2 = 14$
6	$\mathsf{L}^2(6) + \mathsf{L}(2) = 28$
12	$\mathsf{L}^2(12) + \mathsf{L}^2(2) = 56$
24	$\mathsf{L}^2(24) + \mathsf{L}^3(2) = 112$
48	$\mathsf{L}^2(28) + \mathsf{L}^4(2) = 224$
96	$\mathsf{L}^2(96) + \mathsf{L}^5(2) = 193$
192	$\mathsf{L}^2(192) + \mathsf{L}^6(2) = 131$
129	$\mathsf{L}^2(129) + \mathsf{L}^7(2) = 7$

9.3.2 Construction of $(1, 8)$-correlated sequences

Let (s_0, s_1) be any random $2n$-bit value and $\mathcal{K}_{(k_0,k_1)} = \{(k_0, k_1, k_2, k_3) \mid (k_2, k_3) \in \mathbb{F}_2^n \times \mathbb{F}_2^n\}$ be the set of 2^{2n} keys with k_0 and k_1 fixed to some constant value. For $t \geq 6$ and $0 \leq i < 2^n$, define

$$\mathrm{P}(i, t, \mathcal{K}_{(k_0,k_1)}) = \{(k, S_{(k,t)}) \mid k \in \mathcal{K}_{(k_0,k_1)} \text{ and } s_5 = i\}$$

as the set of keys and their corresponding sequences that map s_5 to i.

We start with the simpler case, i.e., $s_5 = 0$. First, we construct $\mathsf{P}(0, 8, \mathcal{K}_{(k_0,k_1)})$ and then show how to construct $\mathsf{P}(i, 8, \mathcal{K}_{(k_0,k_1)})$ from the knowledge of $\mathsf{P}(0, 8, \mathcal{K}_{(k_0,k_1)})$.

9.3.2.1 Construction of $\mathsf{P}(0, 8, \mathcal{K}_{(k_0,k_1)})$

We divide the construction of $\mathsf{P}(0, 8, \mathcal{K}_{(k_0,k_1)})$ into 3 steps, namely 1) Finding $\mathsf{P}(0, 6, \mathcal{K}_{(k_0,k_1)})$, 2) Obtaining $\mathsf{P}(0, 7, \mathcal{K}_{(k_0,k_1)})$ from $\mathsf{P}(0, 6, \mathcal{K}_{(k_0,k_1)})$, and 3) Obtaining $\mathsf{P}(0, 8, \mathcal{K}_{(k_0,k_1)})$ from $\mathsf{P}(0, 7, \mathcal{K}_{(k_0,k_1)})$. For each step, we denote the number of computations of $f_{(a,b,c)}$ by T_{step}.

Step 1: Finding $\mathsf{P}(0, 6, \mathcal{K}_{(k_0,k_1)})$. We note that $\forall k \in \mathcal{K}_{(k_0,k_1)}$, $S_{(k,4)}$ is a constant sequence and requires only 2 computations of $f_{(a,b,c)}$. Hence, finding the keys for which $s_5 = 0$ is equivalent to solving the equation

$$f_{(a,b,c)}(X + k_2) = k_3 + s_3$$

where $X = f_{(a,b,c)}(s_3) + s_2$. We use z-linear segments (Definition 9.4) to solve this equation. As a result, $T_{step_1} = 3 + N_z$. Note that $|\mathsf{P}(0, 6, \mathcal{K}_{(k_0,k_1)})| = 2^n$, as $s_4 = X + k_2$ can take all 2^n distinct values. Thus for all $(k, S_{(k,6)}) \in \mathsf{P}(0, 6, \mathcal{K}_{(k_0,k_1)})$ the pair (k_2, k_3) is unique. Accordingly, let $\mathsf{I}_{(k_0,k_1)} = \{k_3 \mid (k, S_{(k,6)}) \in \mathsf{P}(0, 6, \mathcal{K}_{(k_0,k_1)})\}$, then $|\mathsf{I}_{(k_0,k_1)}| = 2^n$.

Step 2: Obtaining $\mathsf{P}(0, 7, \mathcal{K}_{(k_0,k_1)})$ from $\mathsf{P}(0, 6, \mathcal{K}_{(k_0,k_1)})$. Let $(k, S_{(k,6)}) \in \mathsf{P}(0, 6, \mathcal{K}_{(k_0,k_1)})$ and consider the following relation $s_4 + s_6$. We have

$$\begin{aligned} s_4 + s_6 &= s_4 + f_{(a,b,c)}(s_5) + k_4 \\ &= s_4 + 0 + s_4 + k_4 \\ \implies & s_6 = s_4 + k_4. \end{aligned}$$

Thus $T_{step_2} = 0$.

Step 3: Obtaining $\mathsf{P}(0, 8, \mathcal{K}_{(k_0,k_1)})$ from $\mathsf{P}(0, 7, \mathcal{K}_{(k_0,k_1)})$. For a given $(k, S_{(k,7)})$ in $\mathsf{P}(0, 7, \mathcal{K}_{(k_0,k_1)})$ we compute s_7 as follows.

$$\begin{aligned} s_7 &= f_{(a,b,c)}(s_6) + s_5 + k_5 = f_{(a,b,c)}(s_6) + k_5 \\ &= f_{(a,b,c)}(s'_4) + k_5 \text{ (By step 1)} \\ &= f_{(a,b,c)}(X + k'_2) + k_5 = s'_3 + k'_3 + k_5 \\ &= s_3 + \mathsf{I}_{(k_0,k_1)}[k'_2] + k_5 \text{ (as } s'_3 = s_3) \\ &= s_3 + \mathsf{I}_{(k_0,k_1)}[k_2 + k_4] + k_5 \end{aligned}$$

Note that since $s_6 = X + k'_2 \implies k'_2 = s_6 + X = s_4 + k_4 + X = k_2 + k_4$ ($s_4 + X = k_2$ follows from Step 1). Furthermore $T_{step_3} = 0$.

Corollary 9.1. Given $\mathsf{P}(0, 8, \mathcal{K}_{(k_0,k_1)})$, $\mathsf{I}_{(k_0,k_1)}$ and $k = (k_0, k_1, 0, \mathsf{I}_{(k_0,k_1)}[0])$. Then

$$|\mathrm{CK}(k)| = 2^n - 1$$

We could use a similar construction shown to the one above to get $\mathsf{P}(i, 8, \mathcal{K}_{(k_0,k_1)})$ for $1 \leq i < 2^n$. However, this would require $2^n(3 + N_z)$ computations of $f_{(a,b,c)}$ in total. Next, we show how to reduce this number to $(3 + 2N_z)$.

9.3.2.2 Computing $\mathsf{P}(i, 8, \mathcal{K}_{(k_0,k_1)})$ from $\mathsf{P}(0, 8, \mathcal{K}_{(k_0,k_1)})$

Theorem 9.1. Given $\mathsf{I}_{(k_0,k_1)}$, $k = (k_0, k_1, 0, \mathrm{I}_{(k_0,k_1)}[0])$, $(k, S_{(k,8)}) \in \mathsf{P}(0, 8, \mathcal{K}_{(k_0,k_1)})$ and $X = f_{(a,b,c)}(s_3) + s_2$. Let $1 \leq i < 2^n$ and $\tilde{k} = (k_0, k_1, 0, \mathsf{I}_{(k_0,k_1)}[0] + i)$. Then the following assertions hold.

1. $S_{(k,5)} = S_{(\tilde{k},5)}$
2. $(\tilde{k}, S_{(\tilde{k},6)}) \in \mathsf{P}(i, 6, \mathcal{K}_{(k_0,k_1)})$
3. $\tilde{s_6} = s_3 + \mathsf{I}_{(k_0,k_1)}[X + i] + X + \tilde{k_2} + \tilde{k_4}$

4. $\tilde{s_7} = s_3 + i + \tilde{k_5} + \mathsf{I}_{(k_0,k_1)}[\tilde{s_6} + X]$

5. $|\mathrm{CK}(\bar{k})| = 2^n - 1$

Proof. 1. Since $k_2 = \tilde{k_2} = 0 \implies s_4 = \tilde{s_4} = X \implies S_{(k,5)} = S_{(\tilde{k},5)}$.

2. It is enough to show that $\tilde{s_5} = i$. We have

$$\begin{aligned} \tilde{s_5} &= f_{(a,b,c)}(\tilde{s_4}) + \tilde{s_3} + \tilde{k_3} = f_{(a,b,c)}(s_4) + s_3 + \tilde{k_3} \\ &= \mathsf{I}_{(k_0,k_1)}[0] + s_3 + s_3 + \mathsf{I}_{(k_0,k_1)}[0] + i = i. \end{aligned}$$

3. We compute $\tilde{s_6}$ as follows.

$$\begin{aligned} \tilde{s_6} &= f_{(a,b,c)}(\tilde{s_5}) + \tilde{s_4} + \tilde{k_4} = f_{(a,b,c)}(i) + \tilde{s_4} + \tilde{k_4} \\ &= s_3 + \mathsf{I}_{(k_0,k_1)}[X + i] + X + \tilde{k_2} + \tilde{k_4}. \end{aligned}$$

4. The proof is similar to the assertion 3.

5. Note that for $1 \le j < 2^n$, $(k_0, k_1, j, \mathsf{I}_{(k_0,k_1)}[j]) \in \mathrm{CK}(k) \iff (k_0, k_1, j, \mathsf{I}_{(k_0,k_1)}[j] + i) \in \mathrm{CK}(\bar{k})$. This follows because $s_5 + \bar{s_5} = k_3 + \bar{k_3} \implies k_3 + \bar{k_3} = i$. Thus, $|\mathrm{CK}(\bar{k})| = 2^n - 1$. □

We use Theorem 9.1 together with z-linear segment sets to compute all partitions. A brief comparison of different approaches with the number of computations of $f_{(a,b,c)}$ to obtain $\mathsf{P}(i, 8, \mathcal{K}_{(k_0,k_1)})$ is provided in Table 9.6.

Table 9.6: Comparison of different approaches with the number of computations of $f_{(a,b,c)}$ for 6 out of r rounds

Approach	# computations of $f_{(a,b,c)}$	
	(a,b,c)	
	$(8,1,2)$	$(5,0,1)$
Naive	$2^{64} \times \frac{6}{r}$	$2^{64} \times \frac{6}{r}$
Theorem 9.1 and z-linear segment sets	$2^{32} \times \frac{(3+1818)}{r}$	$2^{32} \times \frac{(3+1596)}{r}$

9.4 Key Recovery Attack on 25-round Simon and Simeck

In this section, we show the key recovery attack procedure on 25-round Simon-32/64 and Simeck-32/64. We note that construction of $(1,8)$-correlated sequences as shown in Section 9.3 is independent of the key scheduling algorithms. Thus, we simply utilize these sequences for 6 encryption and 6 decryption rounds in an MitM attack (Figure 9.4). As a result, we do partial encryption for 12 rounds, starting from round 6 and match the left half of state, i.e., s_{19} at 19-th round.

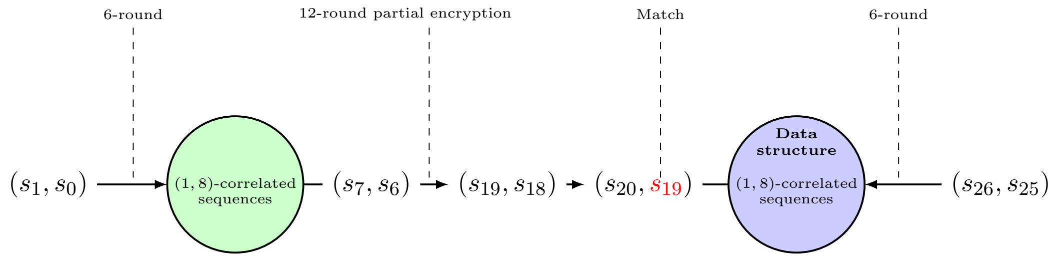


Figure 9.4: 25-round key recovery procedure

From now, we denote s_i^e/s_i^d, k_i^e/k_i^d, X^e/X^d, $\mathsf{I}_{(k_0^e,k_1^e)}/\mathsf{I}_{(k_0^d,k_1^d)}$ and $\mathsf{DS^e}/\mathsf{DS^d}$ as the i-th element of the keyed sequence, i-th subkey, the value of $f_{(a,b,c)}(s_3^e) + s_2^e$ / $f_{(a,b,c)}(s_3^d) + s_2^d$, indexing set and stored data structure from encryption/decryption side, respectively. For example, $s_0^e = s_0$, $s_1^e = s_1$, $s_0^d = s_{26}$, $s_1^d = s_{25}$, $k_{24}^e = k_0^d$ and so on.

In Algorithm 9.1, we present a generic procedure for recovering the secret key. It takes the input as 3 known plaintext-ciphertext pairs encrypted either by Simon-32/64 or Simeck-32/64 and returns the secret key. The attack is divided into two phases, namely 1) Offline phase and 2) Online phase. The time complexities of both phases are given by $T^{offline}$ and T^{online} where a subscript (e.g., T_i^{online}) denotes the time complexity of i-th step of the corresponding phase. In what follows, we present the details of both phases.

9.4.1 Offline phase

In this phase, we first compute z-linear segment sets using Defintion 9.4. Next, we construct a data structure $\mathsf{DS^d}$ that is used in the online phase to compute the value of s_7^d without doing any nonlinear operation. Note that in order to compute s_7^d for any key $k = (k_0^d, k_1^d, k_2^d, k_3^d)$, we only need the values of s_3^d, X^d and $\mathsf{I}_{(k_0^d,k_1^d)}$ (Theorem 9.1). Hence, we store the array $[s_3^d, X^d, \mathsf{I}_{(k_0^d,k_1^d)}]$ as the $(\mathsf{L}^{16}(k_0^d)||k_1^d)$-th row of $\mathsf{DS^d}$ for a fixed (k_0^d, k_1^d) pair.

The procedures compute_zs and construct_ds in Algorithm 9.1 constitute the offline phase. In Appendix B.1, we provide an example of $\mathsf{DS^d}$ for a toy Simon cipher.

Memory complexity. The memory required to store z-linear segment sets in bits is $(N_z + \#\text{ coset leaders}) \times 16$. Furthermore, to store a single row of $\mathsf{DS^d}$, $(1 + 1 + 2^{16}) \times 16$ bit space is needed. Thus the total memory (Mem) is given by

$$\begin{aligned}\mathsf{Mem}_{\mathsf{Simon\text{-}32/64}} &= (N_z + \#\text{ coset leaders}) \times 16 + 2^{32} \times (1 + 1 + 2^{16}) \times 16 \\ &= (909 + 4116) \times 16 + 2^{32} \times (2 + 2^{16}) \times 16 \approx 2^{52} \text{ bits } = 2^{49} \text{ bytes.}\end{aligned}$$

Algorithm 9.1 Key recovery algorithm

```
1: Input : ${(s_0^{e,0}, s_1^{e,0}), (s_0^{d,0}, s_1^{d,0})}, {(s_0^{e,1}, s_1^{e,1}), (s_0^{d,1}, s_1^{d,1})}, {(s_0^{e,2}, s_1^{e,2}), (s_0^{d,2}, s_1^{d,2})}$
2: Output : secret key $k$
3: Procedure main :
4:         // Offline phase                                                    ▷ $T^{offline}$
5:         call procedure compute_zs
6:         call procedure construct_ds
7:         // Online phase                                                     ▷ $T^{online}$
8:         call procedure recover_sk
9:
10: Procedure compute_zs :                                                     ▷ $T_0^{offline}$
11:        // Compute z-linear segment sets using Definition 9.4
12:        $n = 16$, $(a, b, c) = (8, 1, 2)$ / $(5, 0, 1)$
13:        return($Z$, $CL_z$)
14:
15: Procedure construct_ds :                                                   ▷ $T_1^{offline}$
16:        // Construct data structure for 6 decryption rounds
17:        $\mathsf{DS^d} = [\ [\ ]\ ]$
18:        $s_0^d = s_0^{d,0}$, $s_1^d = s_1^{d,0}$
19:        for $k_0^d = 0$ to $2^{16} - 1$ do
20:            for $k_1^d = 0$ to $2^{16} - 1$ do
21:                Compute $s_3^d, X^d, \mathsf{I}_{(k_0^d,k_1^d)}$
22:                DS.append([ $s_3^d, X^d, \mathsf{I}_{(k_0^d,k_1^d)}$ ])
23:            end for
24:        end for
25:        return($\mathsf{DS^d}$)
26:
27: Procedure recover_sk :
28:        // Filtering keys with Algorithm 9.2
29:        $K$ = filter_keys                                                   ▷ $T_0^{online}$
30:        // Exhaustive search on $K$ using second plaintext-ciphertext pair
31:        for $\gamma \in K$ do                                               ▷ $T_1^{online}$
32:            if encryption of $(s_0^{e,1}, s_1^{e,1})$ with $\gamma$ equals $(s_0^{d,1}, s_1^{d,1})$ do
33:                $K_1$.append($\gamma$)
34:            end if
35:        end for
36:        // Exhaustive search on $K_1$ using third plaintext-ciphertext pair
37:        for $\gamma \in K_1$ do                                             ▷ $T_2^{online}$
38:            if encryption of $(s_0^{e,2}, s_1^{e,2})$ with $\gamma$ equals $(s_0^{d,2}, s_1^{d,2})$ do
39:                $K_2$.append($\gamma$)
40:            end if
41:        end for
42:        return($K_2$)                                                       ▷ $K_2 = \{k\}$
```

Similarly, $\mathsf{Mem}_{\mathsf{Simeck\text{-}32/64}} \approx 2^{49}$ bytes as $N_z = 798$ (Table 9.4).

Time complexity. The time complexity in terms of the number of computations of $f_{(a,b,c)}$ is given by

$$\begin{aligned} T^{offline} &= T_0^{offline} + T_1^{offline} \\ &= \#\text{ coset leaders} + 2^{32} \times \underbrace{\left(\frac{3+N_z}{25}\right)}_{\#\text{ computations of } f_{(a,b,c)} \text{ to get } \mathsf{I}_{(k_0^d,k_1^d)}} \end{aligned}$$

Thus, $T^{offline} \approx 2^{37.18}$ and 2^{37} for Simon-32/64 and Simeck-32/64, respectively.

9.4.2 Online phase

In this phase, we recover the secret key that maps the plaintext $(s_0^{e,i}, s_1^{e,i})$ to ciphertext $(s_0^{d,i}, s_1^{d,i})$ for $i = 0, 1, 2$. We first find the key set K that maps $(s_0^{e,0}, s_1^{e,0})$ to $(s_0^{d,0}, s_1^{d,0})$ using filter_keys procedure in Algorithm 9.2. Note that $|K| = 2^{48}$ as partial matching is done at the 19-th round (step 16 of Algorithm 9.2). Next, we perform an exhaustive search on the remaining 2 plaintext-ciphertext pairs to get the correct key (steps 31-41 of Algorithm 9.1). We now present the details of filter_keys procedure.

Procedure filter_keys. For a fixed (k_0^e, k_1^e) pair, we first compute s_3^e, $X^e = f_{(a,b,c)}(s_3^e) + s_2^e$ and the indexing set $\mathsf{I}_{(k_0^e,k_1^e)}$. Then we use Theorem 9.1 and z-linear segment sets to compute partitions $\mathsf{P}(i, 8, \mathcal{K}_{(k_0^e,k_1^e)})$ (steps 7-14 of Algorithm 9.2). Next, we do encryption for 12 rounds and check whether s_{19}^e matches with s_7^d or not. If so, the corresponding key is a possible key candidate. The number of computations of $f_{(a,b,c)}$ is then calculated as follows:

$$\begin{aligned} &\underbrace{\frac{3+N_z}{25}}_{\#\text{ computations of } f_{(a,b,c)} \text{ to get } \mathsf{I}_{(k_0^e,k_1^e)}} + \underbrace{\frac{N_z}{25}}_{\#\text{ computations of } f_{(a,b,c)} \text{ to get } \mathsf{P}(i,8,\mathcal{K}_{(k_0^e,k_1^e)})} \\ + \quad & 2^{32} \times \underbrace{\frac{12}{25}}_{\text{12-round encryption}} = \frac{3+2N_z}{25} + 2^{32} \times \frac{12}{25} \end{aligned}$$

The time complexity (T_0^{online}) of filter_keys then equals $2^{32} \times (\frac{3+2N_z}{25} + 2^{32} \times \frac{12}{25}) \approx 2^{64} \times \frac{12}{25}$. In Appendix B.2, we provide an example of computation of s_7^d from DS^{d}.

Algorithm 9.2 Extracting keys that maps (s_0^e, s_1^e) to (s_0^d, s_1^d)

```
1: $s_0^e = s_0^{e,0}$, $s_1^e = s_1^{e,0}$
2: $\mathcal{K} = []$
3: Procedure filter_keys :
4:         for $k_0^e = 0$ to $2^{16} - 1$ do
5:             for $k_1^e = 0$ to $2^{16} - 1$ do
6:                 Compute $s_3^e$, $X^e = f_{(a,b,c)}(s_3^e) + s_2^e$ and $\mathsf{I}_{(k_0^e,k_1^e)}$
7:                 for $z$ in $z$-linear segment sets do
8:                     for $x \in CL_z$ do
9:                         for $i = 0$ to $|C_x| - 1$
10:                            $f = L^c(C_x[i]) + L^i(z)$
11:                            for $j = 0$ to $2^{16} - 1$ do
12:                                $k = (k_0^e, k_1^e, j, \mathsf{I}_{(k_0^e,k_1^e)}[j] + C_x[i])$
13:                                $s_6^e = f + X^e + k_2^e + k_4^e$
14:                                $s_7^e = s_3^e + k_5^e + C_x[i] + \mathsf{I}_{(k_0^e,k_1^e)}[s_6^e + X^e]$
15:                                Encrypt $(s_7^e, s_6^e)$ for 12 rounds and get $s_{19}^e$
16:                                if $s_{19}^e$==compute_$s_7^d(k, \mathsf{DS^d})$ do
17:                                    $\mathcal{K}$.append$(k)$
18:                                end if
19:                            end for
20:                        end for
21:                    end for
22:                end for
23:            end for
24:        end for
25:        return$(\mathcal{K})$
26:
27: Procedure compute_$s_7^d(k, \mathsf{DS^d})$ :
28:        // Compute 8-th element of sequence from decryption side (Theorem 9.1)
29:        $s_3^d = \mathsf{DS^d}[L^{16}(k_0^d) + k_1^d][0]$
30:        $X^d = \mathsf{DS^d}[L^{16}(k_0^d) + k_1^d][1]$
31:        $\mathsf{I}_{(k_0^d,k_1^d)} = \mathsf{DS^d}[L^{16}(k_0^d) + k_1^d][2]$
32:        $p = \mathsf{I}_{(k_0^d,k_1^d)}[k_2^d + k_3^d]$
33:        $s_6^d = \mathsf{I}_{(k_0^d,k_1^d)}[p + X^d] + s_3^d + X^d + k_2^d + k_4^d$
34:        $s_7^d = s_3^d + k_5^d + p + \mathsf{I}_{(k_0^d,k_1^d)}[s_6^d + X^d]$
35:        return$(s_7^d)$
```

Time complexity. The time complexity of the complete attack is dominated by T^{online} which is given by:

$$\begin{aligned} T^{online} &= T_0^{online} + T_1^{online} + T_2^{online} \\ &= 2^{64} \times \frac{12}{25} + 2^{48} + 2^{16} \approx 2^{62.94} \end{aligned}$$

Remark 9.2. For the 24-round attack, the data and memory complexities are the same. However, the time complexity is $2^{64} \times \frac{11}{24} \approx 2^{62.87}$.

9.4.3 Experimental verification

We have implemented the complete attack in Python. We ran experiments for toy versions of both ciphers, i.e., with blocksize/keysize, 8/16 and 16/32-bit. We have found that the attack works for all keys, and for any 3 distinct random plaintext encrypted either by Simon-8/16 (Simon-16/32) or Simeck-8/16 (Simeck-16/32). Hence, a success probability of 1 implies that the similar results hold for Simon-32/64 and Simeck-32/64.

9.5 Improved Key Recovery Attacks

In this section, we show how to improve the key recovery attack presented in the previous section by 2 rounds with the same complexities as the 25-round attack. For a fixed partition $\mathsf{P}(i, 8, \mathcal{K}_{(k_0^e,k_1^e)})$, we incorporate the properties of key scheduling algorithms (Section 2.5.3) and one round differentials and show that $\mathsf{P}(i, 9, \mathcal{K}_{(k_0^e,k_1^e)})$ can be computed from $\mathsf{P}(i, 8, \mathcal{K}_{(k_0^e,k_1^e)})$ by computing $f_{(a,b,c)}$ at most 2^{15} times. As a result, both forward and middle rounds can be extended by one round each, i.e., partial encryption starts from round 7 and matching is done at the 20-th round. The results of the following two properties can be obtained directly by the definition of $\mathsf{P}(i, 8, \mathcal{K}_{(k_0^e,k_1^e)})$ and the key scheduling algorithm. We present the main result of this section in Lemma 9.1.

Property 9.3 (Simon KSA and $\mathsf{P}(i, 8, \mathcal{K}_{(k_0^e,k_1^e)})$)**.** Let $F : \mathbb{F}_2^{16} \to \mathbb{F}_2^{16}$ be such that $F(x) = f_{(8,1,2)}(x + \Delta_y) + x + L^{15}(x) + L^{10}(y) + L^8(y)$, where $y = \mathsf{I}_{(k_0^e,k_1^e)}[x]$ and $\Delta_y = L^{13}(y) + L^{12}(y)$. Then $|Img(F)| \leq 2^{15}$ where $Img(F)$ is the image set of F.

Property 9.4 (Simeck KSA)**.** Let $n \geq 4$, $k_{i+4}^e = f_{(5,0,1)}(k_{i+1}^e) + k_i^e$ and $i \geq 0$. Then for a fixed (k_0^e, k_1^e) pair, k_4^e is constant for all $2^n \times 2^n$ values of k_2^e and k_3^e.

Property 9.5 (Differential property [86])**.** Let $n \geq 4$, $\Delta \in \mathbb{F}_2^n$ be fixed. Then

$$|Img(f_{(a,b,c)}(x) + f_{(a,b,c)}(x + \Delta))| \leq 2^{n-1}.$$

Lemma 9.1. Given $n = 16$ and $(a, b, c) = (8, 1, 2)/(5, 0, 1)$. Then for all $(k, S_{(k,8)}) \in \mathsf{P}(i, 8, \mathcal{K}_{(k_0^e,k_1^e)})$, s_7^e can take at most 2^{15} values.

Proof. Consider the value of s_7^e in the following cases:

- Case 1 : $(a, b, c) = (8, 1, 2)$

$$\begin{aligned} s_7^e &= f_{(8,1,2)}(s_6^e) + s_5^e + k_5^e = f_{(8,1,2)}(s_4^e + k_4^e + f_{(8,1,2)}(i)) + i + k_5^e \\ &= f_{(8,1,2)}(X^e + k_2^e + k_4^e + f_{(8,1,2)}(i)) + i + k_5^e,\ X^e = f_{(8,1,2)}(s_3^e) + s_2^e \\ &= f_{(8,1,2)}(C_0 + k_2^e + (L^{13}(k_3^e) + L^{12}(k_3^e)) + \\ &\quad C_1 + k_2^e + L^{15}(k_2^e) + L^{10}(k_3^e) + L^8(k_3^e) \ \text{(Simon KSA)} \end{aligned}$$

  Here C_0 and C_1 are constants and given by

$$\begin{aligned} C_0 &= X^e + f_{(8,1,2)}(i) + k_0^e + k_1^e + L^{15}(k_1^e) + Z_0 \\ C_1 &= i + Z_1 + L^{13}(Z_0) + L^{12}(Z_0) + L^{13}(k_0^e) + L^{12}(k_0^e) + \\ &\quad k_1^e + L^{13}(k_1^e) + L^{11}(k_1^e) \end{aligned}$$

  By Property 9.3, s_7^e can take at most 2^{15} values.

- Case 2 : $(a, b, c) = (5, 0, 1)$

$$\begin{aligned} s_7^e &= f_{(5,0,1)}(s_6^e) + s_5^e + k_5^e = f_{(5,0,1)}(s_4^e + k_4^e + f_{(5,0,1)}(i)) + i + k_5^e \\ &= f_{(5,0,1)}(X^e + k_2^e + k_4^e + f_{(5,0,1)}(i)) + i + k_5^e,\ X^e = f_{(5,0,1)}(s_3^e) + s_2^e \\ &= f_{(5,0,1)}(\Delta + k_2^e) + C_1 + f_{(5,0,1)}(k_2^e) \ \text{(Property 9.4)} \end{aligned}$$

  Similar to previous case, Δ and C_1 are constants and given by:

$$\begin{aligned} \Delta &= X^e + f_{(5,0,1)}(i) + k_0^e + f_{(5,0,1)}(k_1^e) + Z_0 \\ C_1 &= i + Z_1 + k_1^e \end{aligned}$$

  The proof then follows from Property 9.5.

□

From Lemma 9.1, we note that for each partition $\mathsf{P}(i, 8, \mathcal{K}_{(k_0^e, k_1^e)})$, s_7^e can take at most 2^{15} values. Accordingly, we only modify steps 11-18 of Algorithm 9.2. The partial encryption starts from (s_8^e, s_7^e) and s_{21}^e is then used for the matching. The modification is presented in Algorithm 9.3.

Algorithm 9.3 Modified algorithm for 27-round key recovery attack

```
1: TEMP_S6 = [ ]
2: TEMP_S7= [ ]
3: TEMP_Uniq_S7 = [ ]
4: for $j = 0$ to $2^{16} - 1$ do
5:         $k = (k_0^e, k_1^e, j, \mathsf{I}_{(k_0^e,k_1^e)}[j] + C_x[i])$
6:         $s_6^e = f + X^e + k_2^e + k_4^e$
7:         $s_7^e = s_3^e + k_5^e + C_x[i] + \mathsf{I}_{(k_0^e,k_1^e)}[s_6^e + X^e]$
8:         TEMP_S6.append($s_6^e$)
9:         TEMP_S7.append($s_7^e$)
10: end for
11:
12: // Unique value of TEMP_S7
13: TEMP_Uniq_S7 = unique(TEMP_S7)
14:
15: for $u$ in TEMP_Uniq_S7 do
16:         $t^e = f_{(a,b,c)}(u)$
17:         // get_index finds indexes $l$ such that TEMP_S7[$l$] = $u$
18:         Indices = get_index(TEMP_S7)
19:             for $ind$ in Indices do
20:                 $k = (k_0^e, k_1^e, ind, \mathsf{I}_{(k_0^e,k_1^e)}[ind] + C_x[i])$
21:                 $s_8^e = t^e + k_6^e$+TEMP_S6[$ind$]
22:                 Encrypt $(s_8^e, u)$ for 13 rounds and get $s_{21}^e$
23:                 if $s_{21}^e$==compute_$s_7^d(k$, DS$^d$) do
24:                     $\mathcal{K}$.append($k$)
25:                 end if
26:             end for
27: end for
```

Attack complexities. The data and memory complexities are the same as the 25-round attack. The time complexity is given by:

$$\begin{aligned}
T^{online} &= 2^{32} \times T_0^{online} + T_1^{online} + T_2^{online} \\
&\approx 2^{32}(\frac{3+2N_z}{27} + 2^{31} \times \frac{1}{27} + 2^{32} \times \frac{13}{27}) + 2^{48} + 2^{16} \\
&\approx 2^{64} \times \frac{13}{27} \approx 2^{62.94}
\end{aligned}$$

Remark 9.3. The complexities of 26-round attack are calculated accordingly.

9.6 Summary

In this chapter, we have introduced a new characteristic of block ciphers called correlated sequences and demonstrated its application in a meet-in-the-middle attack. As a result, we presented a $2t - 3 + l$ (resp. $2t + l$)-round attack for Feistel, i.e., NLFSR (resp. SPN) ciphers with t length correlated sequences and l rounds of partial encryption. We have applied our technique on two lightweight block ciphers Simon-32/64 and Simeck-32/64 and presented the first 24, 25, 26, 27-round attacks on these ciphers with data and memory complexities of 3 and 2^{49} bytes, respectively. The time complexities are $2^{62.87}$(resp. $2^{62.94}$) for 24, 26 (resp. 25, 27)-round attacks.

Chapter 10

Practical Forgery Attacks on Limdolen and HERN

Contents

10.1 Introduction

Limdolen and HERN are round 1 candidates of the NIST Lightweight Cryptography Standardization Project [5]. Limdolen is a family of lightweight AEAD algorithms with key sizes 128 and 256 bits. It adopts a Parallelizable Message Authentication Code (PMAC) [39] mode to compute a tag and then uses counter mode of encryption to generate the ciphertext. However, compared to PMAC where random and indistinguishable secret masks[1] are used, Limdolen-128/(256) utilizes two distinct 128(256)-bit secret masks only. The designers state that "*Due to Limdolen's target of constrained environments, rather than a series of calculations, we will alternate between $i = 0$ and $i = 1$, the two most common values of i in $\gamma^i L$.*" Moreover, during the tag computation phase, *the associated data and message are first combined together to form a single input and then the padding procedure ie executed.* Based on the design choices and security proofs of PMAC and counter mode of encryption, the designers claim 128(256)-bit integrity security for Limdolen-128(256).

[1]masks derived from PMAC key where PMAC key equals $\mathsf{Enc}_K(0^n)$

On the other hand, HERN is a 128-bit authenticated encryption scheme and adopts a stream cipher style construction similar to the CAESAR finalist Acorn [1, 133]. The state size is 256 bits and at each clock cycle, 4 nonlinear bits are fed back to the state (except during ciphertext and tag generation phase). *After processing the associated data, the state is updated 512 times by adding '0' bit stream to the feedback bits. A similar procedure is applied after plaintext processing.* Accordingly, they claim that HERN achieves 128-bit integrity security.

In this chapter, we show that some non-conservative design choices (highlighted in italics above) made by the designers solely to achieve a lightweight design lead to practical forgery attacks. In particular, we show the construction of associated data-only, ciphertext-only and associated data and ciphertext forgeries which require a feasible number of forging attempts.

Outline. The rest of the chapter is organized as follows. A brief description of Limdolen is provided in Section 10.2. In Section 10.3, we present the details of forgery attacks on Limdolen along with the experimental results. Sections 10.4 and 10.5 present the specification and forgery attacks on HERN, respectively. Finally, we conclude in Section 10.6.

10.2 Specification of Limdolen

Limdolen [98] is a family of lightweight AEAD algorithms with key sizes 128 and 256 bits. We denote an instance of Limdolen by Limdolen-n and its corresponding underlying block cipher[2] by Limdolen-BC-n where $n \in \{128, 256\}$. In this section, we first give a brief overview of Limdolen-n and then list the security goals claimed by the designers.

10.2.1 Description of Limdolen AEAD

Limdolen adopts a tweaked PMAC based construction to provide AEAD functionality. It has two variants Limdolen-n, $n \in \{128, 256\}$. For both the variants, the size of key, nonce and tag are equal to n bits. A high level overview of individual phases of Limdolen-n is described below.

Padding. The associated data AD and the message M are first concatenated together to form a single input message. It is then divided into chunks of n-bit blocks, i.e., $(X_0, \cdots, X_{l-1}) \xleftarrow{n} AD||M$. If $|X_{l-1}| = n$, then a single byte is XORed to the last byte of X_{l-1}. This pad_byte equals 0xC0 (0x80) depending on whether the length of associated data is zero (non-zero). In case the number of bytes of X_{l-1} is less than $n/8$, first a pad_byte is appended to X_{l-1}, followed by adding zero bytes until the block length becomes n. This procedure is denoted by addPaddingMarker$(\cdot)$.

[2]Our attack is independent of the block cipher specification and hence the reader is referred to [98] for more details.

Remark 10.1. The padding rule described above follows the Limdolen's specification document. However, in the reference implementation the pad_byte is always XORed to the last byte of X_{l-1}. We emphasize that our attacks are independent of location of this byte.

Tag generation. The tag computation of Limdolen-n is similar to PMAC and is shown in Figure 10.1. First the PMAC key is derived by encrypting nonce N with the master key K. We denote it by aK where aK = Limdolen-BC-$n(K, N)$. Next, three n-bit masks given by

$$\begin{aligned} \alpha &= \text{Limdolen-BC-}n(\text{aK}, 0^n) \\ \text{alpha\_x} &= \text{LB}(\alpha) \\ \text{alpha\_inv\_x} &= \text{RB}(\alpha) \end{aligned}$$

are computed where the function LB(α) (resp. RB(α)) rotates each byte of α left (resp. right) by 1. Each n-bit block X_i (except the last block) is XORed alternately with α or alpha_x which is then encrypted with Limdolen-BC-n using aK as the key. At each iteration, the output is XORed to δ_c which acts as a checksum. The tag is then given by

$$T = \text{Limdolen-BC-}n(\text{aK}, \delta_c \oplus \text{alpha\_inv\_x} \oplus \text{addPaddingMarker}(X_{l-1})).$$

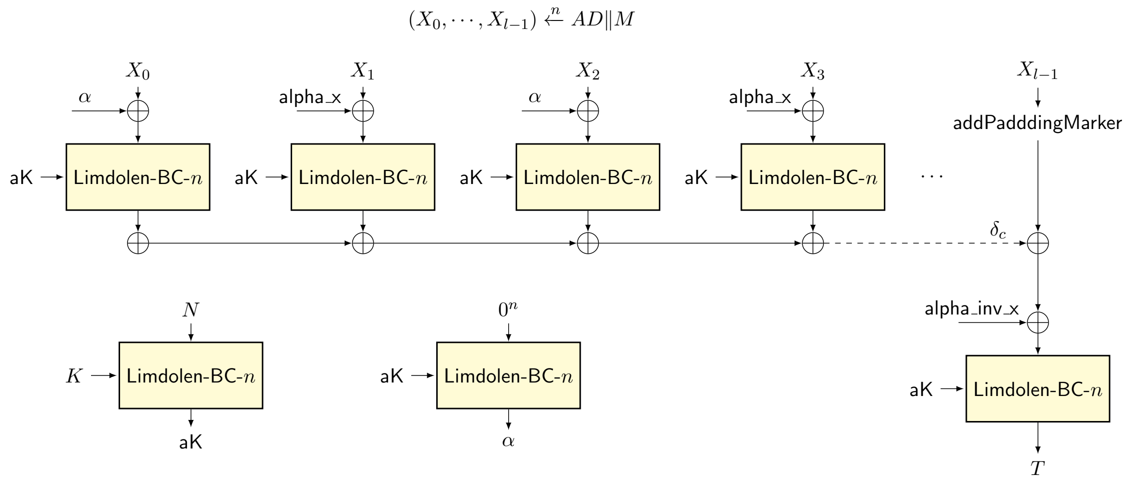


Figure 10.1: Tag generation phase of Limdolen-n

Encryption. The encryption is similar to the counter-mode of operation. The XOR value of nonce and tag is used as the intial counter. This phase is shown in Figure 10.2. The decryption is similar to encryption and hence the details are omitted.

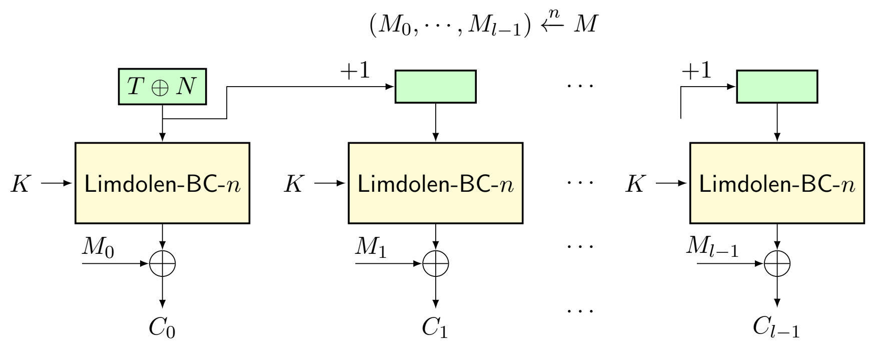


Figure 10.2: Encryption phase of Limdolen-n

10.2.2 Security claims

The security claims of Limdolen in the nonce-respecting setting are summarized in Table 10.1.

Table 10.1: Security claims of Limdolen in bits [98]

Goal	Limdolen-128	Limdolen-256
Confidentiality of plaintext	128	256
Integrity of plaintext	128	256
Integrity of associated data	128	256
Data limit (in blocks)	2^{64}	2^{128}

10.3 Forgery Attacks on Limdolen

In this section, we present the details of forgery attacks on both variants of Limdolen. First, we give a brief overview of the adversarial model and the main idea of our attack. Next, we show the construction of associated data-only, ciphertext-only and associated data and ciphertext forgeries that require a single encryption query and one forging attempt for successful verification. Finally, we provide the experimental results.

10.3.1 Adversarial model

We assume that the adversary $\mathcal{A}$ is nonce-respecting, which means it never makes two queries to the encryption oracle with the same nonce. Nevertheless, $\mathcal{A}$ is allowed to repeat nonces in decryption queries. We say that **"$\mathcal{A}$ forges"** if the decryption oracle ever returns a plaintext other than error symbol $\perp$ on input of (N, AD, C, T) where (C, T) has never been output by encryption oracle on input of a query (N, AD, M) for some AD and M [113].

In the sequel, we classify three types of forgeries based on the input modification.

- associated data-only: **"$\mathcal{A}$ forges"** by changing AD and/or T
- ciphertext-only: **"$\mathcal{A}$ forges"** by changing C and/or T
- associated data and ciphertext: **"$\mathcal{A}$ forges"** by changing both AD and C, and/or T.

10.3.2 Core idea of forgery

For simplicity, we explain the idea for a single complete block of associated data which is given in Lemma 10.1.

Lemma 10.1. Let $K \xleftarrow{\$} \{0,1\}^n$ be fixed. Let $N \xleftarrow{\$} \{0,1\}^n$, $AD_0 \xleftarrow{\$} \{0,1\}^n$, $M = \epsilon$ and (ϵ, T) be the corresponding ciphertext and tag pair. Then for a positive integer $i \geq 1$ and $AD'_0 \xleftarrow{\$} \{0,1\}^n$, $AD'_1 \xleftarrow{\$} \{0,1\}^n$ and $AD' = (AD'_0 \| AD'_1 \| AD'_0 \| AD'_1)^i \| AD_0$, we have $C' = \epsilon$ and $T' = T$.

Proof. Since $M' = M = \epsilon \implies C' = C = \epsilon$. We now look at the tag generation of AD and AD'. The respective tags are given by

$$T = \text{Limdolen-BC-}n(\text{aK}, \text{alpha\_inv\^{}x} \ \oplus \text{addPaddingMarker}(AD_0))$$
$$T' = \text{Limdolen-BC-}n(\text{aK}, \delta'_c \oplus \text{alpha\_inv\_x} \ \oplus \text{addPaddingMarker}(AD_0)),$$

where $\delta'_c = 0^n$ ($i = 1$ case is shown in Figure 10.3 for). Thus $T' = T$. $\square$

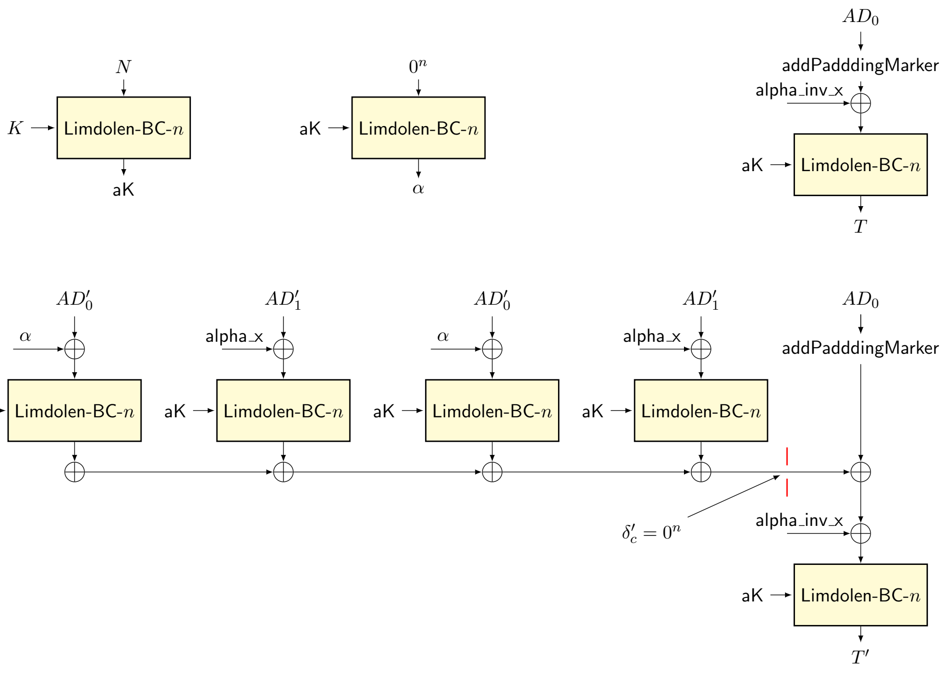


Figure 10.3: Limdolen forgery for a single AD block

Corollary 10.1. To construct forgery for arbitrary number of blocks, we only need to ensure that the XOR sum δ'_c (Figure 10.3) before the last call of block cipher is a constant.

Remark 10.2. Lemma 10.1 trivially holds for partial last block.

In what follows, we describe the basic minimal example of the forgery attack against $\mathsf{Limdolen}\text{-}n$. We assume that blocks are complete and the number of blocks is at least 1. From now onwards, we refer $\mathsf{Limdolen\text{-}BC}\text{-}n$ with key K by $\mathcal{E}^n_K(\cdot)$.

10.3.3 Associated data-only forgery

Let $u \geq 1$ and $i \geq 1$ be two positive integers. Fix $K \xleftarrow{\$} \{0,1\}^n$. We construct forgery as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^n$, $AD \leftarrow \{0,1\}^{u\times n}$, $(AD_0, \cdots, AD_{u-1}) \xleftarrow{n} AD$ and $M = \epsilon$. Encrypt (N, AD, M) and observe (C, T).

Step 2 Let $X, Y \xleftarrow{\$} \{0,1\}^n$ and $W = X\|Y\|X\|Y$.

Step 3 Forge with (N, AD', C, T) where

$$AD' = AD_0\|\cdots\|AD_{u-2}\|W^i\|AD_{u-1}.$$

Note that $AD' \neq AD \implies$ the decryption query is valid. This will pass the verification with probability 1 and returns empty plaintext as the output.

Correctness. To see why this forgery works, consider the values of δ_c and δ'_c, which are given by

$$\delta_c = \bigoplus_{i \bmod 2=0}^{i<u-1} \mathcal{E}^n_{\mathsf{aK}}(AD_i \oplus \alpha) \bigoplus_{i \bmod 2=1}^{i<u-1} \mathcal{E}^n_{\mathsf{aK}}(AD_i \oplus \mathsf{alpha\_x})$$

If $u-1$ is even then

$$\begin{aligned}\delta'_c = &\bigoplus_{i \bmod 2=0}^{i<u-1} \mathcal{E}^n_{\mathsf{aK}}(AD_i \oplus \alpha) \bigoplus_{i \bmod 2=1}^{i<u-1} \mathcal{E}^n_{\mathsf{aK}}(AD_i \oplus \mathsf{alpha\_x}) \\ &2i \bigoplus (\mathcal{E}^n_{\mathsf{aK}}(X \oplus \alpha) \oplus \mathcal{E}^n_{\mathsf{aK}}(Y \oplus \mathsf{alpha\_x})) \\ &= \delta_c \oplus 0^n \implies T' = T.\end{aligned}$$

Similarly, if u is odd then $\delta'_c = \delta_c \oplus 0^n$ and $T' = T$. The only difference is that masks α and $\mathsf{alpha\_x}$ are interchanged.

Some observations on associated data-only forgery.

1. The converse also holds true, i.e., given $AD = AD_0 \| \cdots \| AD_{u-2} \| W^i \| AD_{u-1}$, the modified associated data of the form $AD_0 \| \cdots \| AD_{u-2} \| W^l \| AD_{u-1}$ will give the same tag for all l satisfying $1 \le l < i$.

2. The forgery is independent of whether the last block is a partial AD/M block or consists of both AD and M bytes.

3. We can modify AD in a number of ways. For instance, the following modification also results in a successful forgery.

$$AD' = \begin{cases} X \| Y \| AD_0 \| \cdots \| AD_{u-2} \| X \| Y \| AD_{u-1} & \text{if } u \text{ is odd,} \\ Y \| X \| AD_0 \| \cdots \| AD_{u-2} \| X \| Y \| AD_{u-1} & \text{o.w.} \end{cases}$$

10.3.4 Ciphertext-only forgery

Fix an integer $u \ge 4$ and $K \xleftarrow{\$} \{0,1\}^n$. Let $S_e = \{0, 2, \cdots, \}$ and $S_o = \{1, 3, \cdots, \}$ be the set of even and odd integers less than $u - 1$. Consider two permutations π and ψ which permute the sets S_e and S_o, respectively. Assume that π and ψ are not identity permutations simultaneously. We now construct a forgery as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^n$, $AD = \epsilon$, $M \xleftarrow{n} \{0,1\}^{u \times n}$ and $(M_0, \cdots, M_{u-1}) \xleftarrow{n} M$. Encrypt (N, AD, M) and observe (C, T).

Step 2 Let $(C_0, \cdots, C_{u-2}, C_{u-1}) \xleftarrow{n} C$ and compute $Z_i = M_i \oplus C_i$ for $i = 0, \cdots, u-2$.

Step 3 Forge with (N, AD, C', T) where

$$C' = Z_0 \oplus M_{\pi(0)} \| Z_1 \oplus M_{\psi(0)} \| Z_2 \oplus M_{\pi(1)} \| Z_3 \oplus M_{\psi(1)} \| \cdots \| C_{l-1}.$$

We have $C' \neq C \implies$ the decryption query is valid. This will always pass the verification and returns

$$M_{\pi(0)} \| M_{\psi(0)} \| M_{\pi(1)} \| M_{\psi(1)} \| \cdots \| M_{l-1}$$

as the output.

Correctness. To see the correctness of this forgery, consider the decryption of (N, AD, C', T). First note that the ciphertext computation is done via counter mode of operation (Figure 10.2). Since the counter $T \oplus N$ is same for both encryption and decryption queries, then $M' = M_{\pi(0)} \| M_{\psi(0)} \| M_{\pi(1)} \| M_{\psi(1)} \| \cdots \| M_{l-1}$ is obtained (not released yet). Next, to see if the tags of M' and M are same it is enough to show that $\delta'_c = \delta_c$. This follows trivially as the masking value is α and $\mathsf{alpha\_x}$ for each element in S_e and S_o, respectively. So, permutating these sets individually will not change the XOR sum value. Formally, we have

$$\begin{aligned}\delta'_c &= \bigoplus_{\pi(i), i\in S_e} \mathcal{E}^n_{\mathsf{aK}}(M_{\pi(i)} \oplus \alpha) \bigoplus_{\psi(i), i\in S_o} \mathcal{E}^n_{\mathsf{aK}}(M_{\psi(i)} \oplus \mathsf{alpha\_x}) \\ &= \bigoplus_{i\in S_e} \mathcal{E}^n_{\mathsf{aK}}(M_i \oplus \alpha) \bigoplus_{i\in S_o} \mathcal{E}^n_{\mathsf{aK}}(M_i \oplus \mathsf{alpha\_x}) \\ &= \delta_c \implies T' = T.\end{aligned}$$

Remark 10.3. If π and ψ both are identity permutations then $C' = C \implies$ the decryption query is not valid. The number of valid forgeries then equals $\lceil \frac{u}{2} \rceil \lceil \frac{u-1}{2} \rceil - 1$. Furthermore, these are independent of the length of the last message block.

Remark 10.4. Associated data and ciphertext forgery is a direct application of associated data-only and ciphertext-only forgeries.

10.3.5 Forgeries sssociated with last block

Until now, we have considered the cases where the last block is not modified. To forge the last block, all the previous blocks before it must contain AD bytes. Assume there is only 1 block and it consists of u bytes of AD and v bytes of M such that $u + v \le n/8$. The forgery then proceeds as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^n$. Encrypt (N, AD, M) and observe (C, T).

Step 2 Compute the keystream bytes $Z[i] = M[i] \oplus C[i]$ for $i = 0, \cdots, v-1$

Step 3 For $1 \le l \le v$, forge with (N, AD', C', T) where $AD' = AD \| M[0] \| M[l-1]$ and

$$C' = \begin{cases} \epsilon & \text{if } l = v, \\ Z[0] \oplus M[l] \| \cdots \| Z[v-l-1] \oplus M[v-1] & \text{o.w.} \end{cases}$$

We have $AD' \ne AD$ and $C' \ne C$. Thus, the decryption query is valid and will pass the verification with probability 1 as $AD' \| M' = AD \| M$. The output is $M' = M[l] \| \cdots \| M[v-1]$. Further note that this is a special case of associated data and ciphertext forgery.

Remark 10.5. The above forgery incorporates both cases of Remark 10.1 whether $\mathsf{pad\_byte}$ is XORed to the last byte of block or it is appended after AD and M bytes in case of $u+v < n/8$.

10.3.6 Experimental verification

We have verified the attacks using the reference implementation of Limdolen. In Tables 10.2 and 10.3, we list the examples of forgeries for Limdolen-128 and Limdolen-256, respectively.

Table 10.2: Examples of forgeries for Limdolen-128

Input data	**associated data-only**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	6B22729F7CEA8F9E1EDFB968365BF23B	6B22729F7CEA8F9E1EDFB968365BF23B
AD	BE0A1CDB4142106B5F2BB5BC8911E75E	A5687AF34938ED433536D8AB281FED78
		5D1808F6DDD8D60B23EE9E0E061A5B93
		A5687AF34938ED433536D8AB281FED78
		5D1808F6DDD8D60B23EE9E0E061A5B93
		BE0A1CDB4142106B5F2BB5BC8911E75E
M	Empty string	Empty string
C	Empty string	Empty string
T	EF4F60E08694CABB285D3841C433645D	EF4F60E08694CABB285D3841C433645D

Input data	**ciphertext-only**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	92C2A61831DCDE2EF3DB6060DF03DD0A	92C2A61831DCDE2EF3DB6060DF03DD0A
AD	Empty string	Empty string
M	ACCC9952DBB1CC0C8FA8106D463F483A	19B86CF46A3800F9E01066264FAF600E
	BF23441F82A4BC61D2BF42AF6E4C1F1A	BF23441F82A4BC61D2BF42AF6E4C1F1A
	19B86CF46A3800F9E01066264FAF600E	ACCC9952DBB1CC0C8FA8106D463F483A
	D2A42D5449E9B51BA9F8CB1744EA315D	D2A42D5449E9B51BA9F8CB1744EA315D
C	07AC6C25FAF2BA41F3B808502BA15F66	B2D899834B7B76B49C007E1B22317752
	13237F247E2777389835C8C5B88BC655	13237F247E2777389835C8C5B88BC655
	E5EB9286DF5EE3FB8140B3588BC18C11	509F67206ED72F0EEEF8C5138251A425
	FBF38906197E5B6E069E50E4D8FABF45	FBF38906197E5B6E069E50E4D8FABF45
T	EDFDDE9B652A0FB16A7BFF22FD3B44D8	EDFDDE9B652A0FB16A7BFF22FD3B44D8

Input data	**associated data and ciphertext**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	2B2CC56156A6ACF4D3B1CCE369F4C934	2B2CC56156A6ACF4D3B1CCE369F4C934
AD	0C558F14C1E88FED	0C558F14C1E88FED60D1B7E5BA6EDC
M	60D1B7E5BA6EDC62	62
CT	93C6C56CBBF3B39D	91
T	C248D7D75062DE6163AFC13CADEBC55B	C248D7D75062DE6163AFC13CADEBC55B

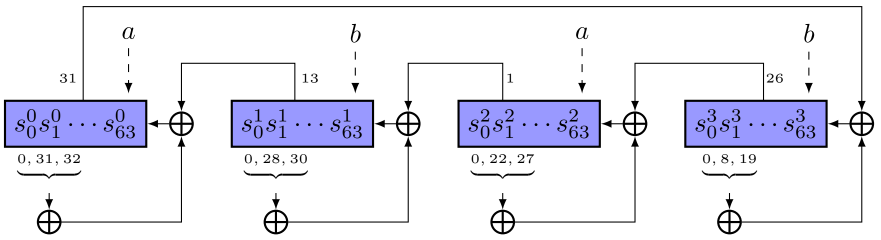
10.4 Specification of HERN

HERN adopts a stream cipher based construction similar to the CAESAR finalist Acorn [133]. The state consists of four 64-bit registers which are updated in an LFSR based style by feeding

Table 10.3: Examples of forgeries for Limdolen-256

Input data	**associated data-only**	
K	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
N	F1C79DD92DA67B984480270726EAB7568B4F1AA10C3BB0B525549E4239265B99	F1C79DD92DA67B984480270726EAB7568B4F1AA10C3BB0B525549E4239265B99
AD	5DA7FC78E3F3692D526069F6DD622EA81E2929484787D3F4354C5CC42DF07CE6	9A0F11FDF7A50B9B8F7C4CF1EB76932DF7E3ED26188C255317E18DE9E9BF6EAB E8B5B01D38A75A30F02DBE8517460F2E3C09E0E4CB2327B4CF63D2795F7DEC65 9A0F11FDF7A50B9B8F7C4CF1EB76932DF7E3ED26188C255317E18DE9E9BF6EAB E8B5B01D38A75A30F02DBE8517460F2E3C09E0E4CB2327B4CF63D2795F7DEC65 5DA7FC78E3F3692D526069F6DD622EA81E2929484787D3F4354C5CC42DF07CE6
M	Empty string	Empty string
C	Empty string	Empty string
T	301A471671BDF1CFAE68714DE61562000F8012DA449F8562E58B7635DC819CAC	301A471671BDF1CFAE68714DE61562000F8012DA449F8562E58B7635DC819CAC
Input data	**ciphertext-only**	
K	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
N	8196CF5D26A4D3728EC8D8B2CA5CA01EF7394366A2A98A09EA6CE9FBF3CCAAB5	8196CF5D26A4D3728EC8D8B2CA5CA01EF7394366A2A98A09EA6CE9FBF3CCAAB5
AD	Empty string	Empty string
M	9EEE67E185CE4A27D8F49C630FA67BF978E7BB6106B714F90FE08CB9CA425A68 30C149B58F94DC688879CB971F4691972E4CF834030C2D12EDB9CBB7FB25202C E769E176FDBEDE8537A91D56F0AEED1EFAE552FEF17F10DE38DC963401B660E8 1F415F1DFF3DA236E7BF8CD76D79F5685E476650C6762EFE52C432547A923C9A	E769E176FDBEDE8537A91D56F0AEED1EFAE552FEF17F10DE38DC963401B660E8 30C149B58F94DC688879CB971F4691972E4CF834030C2D12EDB9CBB7FB25202C 9EEE67E185CE4A27D8F49C630FA67BF978E7BB6106B714F90FE08CB9CA425A68 1F415F1DFF3DA236E7BF8CD76D79F5685E476650C6762EFE52C432547A923C9A
C	DF35A5881ADE06A920E381ADC2DE31A12E33E72C969EE55F35BF7DE2955FE1A1 4462C84E15647050EFDFC01B37FEBC0A0AC1EE3E02BED877CC233A9C2FE38900 2086D28CD3FF11D08F27CFE769BE4C914806A3DAE1676EFC7CC3135A508CA7E3 9CEE6811416763C0AA2A012395D883F5C2C9FC12EDDBCB509381739F0A9738EA	A6B2231F62AE920BCFBE00983DD6A746AC310EB36156E1780283676F5EABDB21 4462C84E15647050EFDFC01B37FEBC0A0AC1EE3E02BED877CC233A9C2FE38900 5901541BAB8F8572607A4ED296B6DA76CA044A4516AF6ADB4BFF09D79B789D63 9CEE6811416763C0AA2A012395D883F5C2C9FC12EDDBCB509381739F0A9738EA
T	3B4230CF23BB7D7E413E13451E8B899856A45A9C7ECB77FF32F257C7BD8780DA	3B4230CF23BB7D7E413E13451E8B899856A45A9C7ECB77FF32F257C7BD8780DA
Input data	**associated data and ciphertext**	
K	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F	000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F
N	7C5734DCCA90853A2959276055D75ABDD4A0AD9BA48B4A845BD99D935FFDA78F	7C5734DCCA90853A2959276055D75ABDD4A0AD9BA48B4A845BD99D935FFDA78F
AD	CF84ACF34B794508DA221B691F332141	CF84ACF34B794508DA221B691F3321412F7C1F507F95FDA0E177B57A66C6C2
M	2F7C1F507F95FDA0E177B57A66C6C2F0	F0
C	8341876562C3BF87B49A155858082690	5C
T	5D8D4291C38C8FC922D7B697E873860593FD26971E590710D30A1F348A41E665	5D8D4291C38C8FC922D7B697E873860593FD26971E590710D30A1F348A41E665

the two nonlinearly generated bits a and b to the registers. A pictorial representation of the HERN state update function is shown in Figure 10.4 and the individual core components are illustrated in Algorithm 10.1.



a
b
a
b
31
13
1
26
$s^0_0 s^0_1 \cdots s^0_{63}$
$s^1_0 s^1_1 \cdots s^1_{63}$
$s^2_0 s^2_1 \cdots s^2_{63}$
$s^3_0 s^3_1 \cdots s^3_{63}$
0, 31, 32
0, 28, 30
0, 22, 27
0, 8, 19

Figure 10.4: Schematic of HERN state update function

Algorithm 10.1 Core components of HERN

```
1: function H_core_step:
2:     $a \leftarrow \mathsf{SB}(s^0_{30}, s^0_{29}, s^1_{32}, s^1_{24}, s^2_{31}, s^2_4, s^3_{15}, s^3_{14})$
3:     $b \leftarrow \mathsf{SB'}(s^0_{30}, s^0_{29}, s^1_{32}, s^1_{24}, s^2_{31}, s^2_4, s^3_{15}, s^3_{14}) \oplus s^0_{32}$
4:     $f^0 \leftarrow s^0_0 \oplus s^0_{31} \oplus s^0_{32} \oplus s^1_{13}$
5:     $f^1 \leftarrow s^1_0 \oplus s^1_{28} \oplus s^1_{30} \oplus s^2_1$
6:     $f^2 \leftarrow s^2_0 \oplus s^2_{22} \oplus s^2_{27} \oplus s^3_{26}$
7:     $f^3 \leftarrow s^3_0 \oplus s^3_8 \oplus s^3_{19} \oplus s^0_{31}$
8:     $s^i_j \leftarrow s^i_{j+1}$, for $i = 0, 1, 2, 3$ and $j = 0, \cdots, 62$
9:     $s^i_{63} \leftarrow f^i$, for $i = 0, 1, 2, 3$

10: function SB($x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3$):
11:     return $1 \oplus x_0y_0 \oplus x_1y_1 \oplus x_2y_2 \oplus x_3y_3$

12: function SB'($x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3$):
13:     return $x_0y_2 \oplus y_0y_3 \oplus x_1x_3 \oplus y_1x_2$
```

```
1: function Adda:
2:     $s^0_{63} \leftarrow s^0_{63} \oplus a$
3:     $s^2_{63} \leftarrow s^2_{63} \oplus a$

4: function Addb:
5:     $s^1_{63} \leftarrow s^1_{63} \oplus b$
6:     $s^3_{63} \leftarrow s^3_{63} \oplus b$

7: function H_if_step($x$):
8:     H_core_step
9:     $a \leftarrow a \oplus x$
10:    Adda
11:    Addb

12: function H_enc_step($m$):
13:    H_core_step
14:    $a \leftarrow a \oplus m$
15:    Adda
16:    $c \leftarrow b \oplus m$
17:    return $c$
```

10.4.1 Description of HERN AEAD

The HERN AEAD algorithm takes as input a 128-bit key K, 128-bit nonce N, *adlen* bits associated data AD, *mlen* bits plaintext M and outputs a *mlen* bits ciphertext C and 128-bit authentication tag T. The encryption consists of 3 phases, namely 1) Initialization, 2) Processing plaintext and 3) Finalization, which are described as follows.

Initialization. The initialization consists of loading the key K and constants into the state and processing the nonce N, associated data AD and running H_if_step (Algorithm 10.1) for 512 steps with zero input.

- Load the state with K and constants. We refer the reader to [137] for more details as this part is irrelevant for our attack.
- Process $N = n_0, n_1, \ldots, n_{127}$. At each step, one bit of N is used to update the state, i.e., H_if_step(n_i), for $i = 0, \cdots, 127$.
- Process $AD = ad_0, ad_1, \ldots, ad_{adlen-1}$. At each step, one bit of AD is used to update the state, i.e., H_if_step(ad_i), for $i = 0, \cdots, adlen - 1$.
- Run the H_if_step for 512 steps with zero-stream, i.e., H_if_step(0), for $i = 0, \cdots, 511$.

Processing plaintext. The plaintext $M = m_0, m_1, \cdots, m_{mlen-1}$ is used to update the state bit-by-bit and the corresponding ciphertext bit is generated using the function $\mathsf{H\_enc\_step}(\cdot)$ (Algorithm 10.1).

- $C \leftarrow \epsilon$
- $c_i \leftarrow \mathsf{H\_enc\_step}(m_i)$, $C \leftarrow C \| c_i$, for $i = 0, \cdots, mlen - 1$

Finalization. After processing all the plaintext bits, the $\mathsf{H\_if\_step}$ runs for 512 times with zero input, and then the tag is generated.

- $\mathsf{H\_if\_step}(0)$, for $i = 0, \cdots, 511$.
- $T \leftarrow \epsilon$
- $t_i \leftarrow \mathsf{H\_enc\_step}(0)$, $T \leftarrow T \| t_i$, for $i = 0, \cdots, mlen - 1$
- return (C, T)

The decryption procedure is identical to encryption.

10.4.2 Security claims

The designers state that *"HERN is designed to have confidentiality of the plaintexts under adaptive chosen-plaintext attacks and the integrity of the ciphertexts under adaptive forgery attacks."* Considering the nonce-respecting setting and a data limit of 2^{64} bits (i.e., $adlen + mlen \leq 2^{64}$), they claim 128-bit security for confidentiality and integrity.

10.5 Forgery Attacks on HERN

In this section, we provide the details of forgery attacks on HERN. In particular, we show that a message can be modified by appending or removing a sequence of consecutive '0' bits of length n. Moreover, we show that the best success rate of forgery is achieved for $n = 1$ case.

The adversarial model is similar to Section 10.3.1. In the following, we explain the minimal example of our forgery attack against HERN. For the description of forgeries, we let S_i, a_i, b_i denote the state of HERN and two nonlinearly generated bits a and b at the beginning of the i-th round.

10.5.1 Associated data-only forgery

Let $1 \leq n \leq 63$ and $K \overset{\$}{\leftarrow} \{0,1\}^{128}$ be fixed. To construct the forgery we proceed as follows.

Step 1 Let $N \overset{\$}{\leftarrow} \{0,1\}^{128}$, $AD \overset{\$}{\leftarrow} \{0,1\}^{\star}$ and $M = \epsilon$. Encrypt (N, AD, M) and observe (C, T).

Step 2 Repeat Step 1 until we obtain a tag whose first n bits are all zero. Define this query as $Q :\overset{\text{def}}{=} (N, AD, M, C, T)$.

Step 3 For each $i = 0$ to $2^n - 1$, decrypt (N', AD', C', T') where

$$N' = N,\ AD' = AD\|0^n,\ C' = \epsilon$$
$$T' = T \ll n \mid (i_0\|\cdots\|i_{n-1}),\ \text{and}\ (i_0, \cdots, i_{n-1}) \xleftarrow{1} i.$$

If the verification succeeds with output as an empty plaintext, we stop.

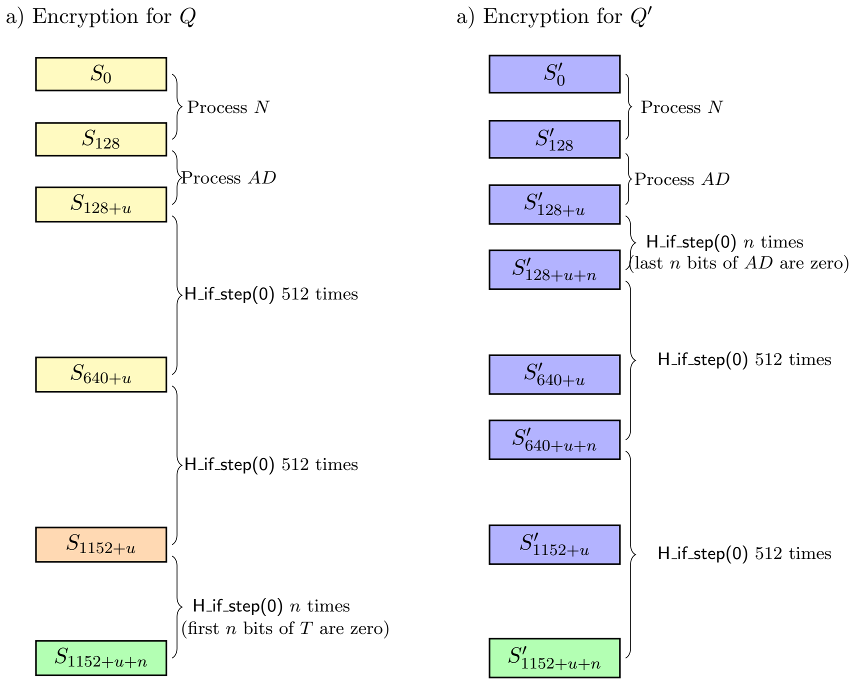


Figure 10.5: Associated data-only forgery of HERN

The decryption queries are valid as $AD' \neq AD$ and $T' \neq T$. To see why such a query work, consider the encryption of Q and $Q' \overset{\text{def}}{=} (N, AD', \epsilon)$. This is illustrated in Lemma 10.2 (also shown in Figure 10.5).

Lemma 10.2. Let Q and Q' be defined as above and $|AD| = u$. Then $T' = T \ll n \mid \Delta$ where Δ is an n-bit string.

Proof. After processing 128 bits of nonce and the first u bits of AD, the states are same, i.e., $S_{128+u} = S'_{128+u}$. For query Q, as M is empty, $\mathsf{H\_if\_step}(\cdot)$ runs for 1024 times with zero input. For Q', since $AD' = AD\|0^n$ and $M' = \epsilon$, $\mathsf{H\_if\_step}(\cdot)$ is iterated for $n + 1024$ times with zero bit. The tag generation phase for Q and Q' starts from S_{1152+u} and $S'_{1152+u+n}$, respectively.

Note that the first n bits of T are zero and they are not added to the state. This is equivalent to the fact that $\mathsf{H\_if\_step}(0)$ runs for another n times starting from round $1152 + u$.

Hence, $S_{1152+u+n} = S'_{1152+u+n} \implies$ the last $128 - n$ bits of T are the same as the first $128-n$ bits of T'. Since the states are unknown, the last n bits of T' has to be guessed. Thus, $T' = T \ll n \mid \Delta$. $\square$

Attack complexities. On average Step 2 requires 2^n encryption queries while Step 3 needs 2^n decryption queries. Thus, for $1 \le n \le 63$, the success rate of forgery is 2^{-n}. For $n = 1$ the success rate is 2^{-1} after querying the encryption oracle 2 times. This clearly violates the designers claim that success rate of forgery is 2^{-127} after two encryption queries.

Some observations on associated data-only forgery.

1. The designers imposed a data limit of 2^{64} bits before a re-keying is done. In order to satisfy this constraint, we restrict the values of n in the range $1, \cdots, 63$. However, this is just a theoretical reasoning and we do not need so many queries especially when we can construct forgery for the $n = 1$ case.
2. The forgery still works if we change 512 to some other number. Hence, it is independent of the number of rounds.

10.5.2 Ciphertext-only forgery

Let $1 \le n \le 31$ and $K \xleftarrow{\$} \{0,1\}^{128}$ be fixed. We construct forgery a as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^{128}$, $AD \xleftarrow{\$} \{0,1\}^{\star}$, $M \leftarrow \{0,1\}^{\ge 1}$. Encrypt $(N, AD, M\|0^n)$ and observe (C, T).

Step 2 Repeat Step 1 until a ciphertext whose last n bits are zero is obtained. Denote this query by (N, AD, M, C, T).

Step 3 Decrypt (N', AD', C', T') where

$$\begin{aligned} N' &= N,\ AD' = AD \\ C' &= c_0\|\cdots\|c_{|M|-n-1} \\ T' &= 0^n|T \gg n. \end{aligned}$$

Step 4 If verification fails, repeat Step 2 and Step 3.

We have $C' \ne C$ as the lengths are different and $T' \ne T$. Thus, each query in step 3 is a valid decryption query. Upon successful verification, only the first $|M|-n$ bits of M are returned. A formal proof of correctness of the decryption query is given in Lemma 10.3.

Lemma 10.3. Let $Q \overset{\text{def}}{:=} (N, AD, M)$ satisy Step 2 with output as (C, T). Let $AD' = AD$, $M' = m_0\|\cdots\|m_{|M|-n-1}$ and $Q' \overset{\text{def}}{:=} (N, AD', M')$. Then $T' = 0^n \mid T \gg n$ iff the bits $b_{1152+|AD|+|M|-n}, \cdots, b_{1152+|AD|+|M|-1}$ are all zero.

Proof. We have $AD' = AD$ and $m'_i = m_i \implies c'_i = c_i$, for $0 \leq i \leq |M| - n - 1$. Therefore, $S_{1152+u+|M|-n} = S'_{1152+u+|M|-n}$. However, the tag generation phase for Q starts from $S_{640+u+|M|}$, and for Q' it starts from $S'_{640+u+|M|-n}$. The corresponding tag bits are given by:

$$t_i = b_{1152+|AD|+|M|+i}$$
$$t'_i = b'_{1152+|AD|+|M|-n+i}.$$

Now, the last n bits of both M and C being zero $\implies S_{1152+|AD|+|M|-n} = S'_{1152+|AD|+|M|-n}$. So, given $b_{1152+|AD|+|M|-n}, \cdots, b_{1152+|AD|+|M|-1}$ are all zero, then $T' = 0^n \mid T \gg n$.

$\square$

Attack complexities. Step 2 requires 2^n encryption queries (on average), while to satisy both Step 2 and Step 3 simultaneously, 2^{2n} encryption queries (on average) are needed. Thus, for $1 \leq n \leq 31$, the success rate of forgery is 2^{-n} after observing the output of 2^{2n} encryption queries. The value of n is chosen to satisfy the data limit restriction of 2^{64} bits.

Remark 10.6. Similar to the associated data-only forgery, the best success rate is achieved for the $n = 1$ case which is 2^{-1} after 4 encryption queries.

10.5.3 Associated data and ciphertext forgery

Let $1 \leq n \leq 63$ and $K \xleftarrow{\$} \{0,1\}^{128}$ be fixed. The forgery then proceeds as follows.

Step 1 Let $N \xleftarrow{\$} \{0,1\}^{128}$, $AD \xleftarrow{\$} \{0,1\}^{\star}$, $M = 0^n$. Encrypt (N, AD, M) and observe (C, T).

Step 2 Repeat step 1 until we obtain $C = 0^n$. Denote this query by (N, AD, M, C, T).

Step 3 Forge with (N', AD', C', T') where

$$N' = N, \; AD' = AD \| 0^n, \; C' = \epsilon, \; \text{and } T' = T,$$

which will always be successful (with empty message as an output) as the states after $640 + |AD| + n$ rounds are the same. The proof is similar to Lemma 10.2.

Attack complexities. Step 2 requires 2^n encryption queries on average, while Step 3 requires only a single decryption query. Thus, for $1 \leq n \leq 63$, the success rate of forgery is 1.

10.5.4 Experimental verification

We have verified the attacks using the reference implementation of HERN [137]. In Table 10.4, we list the examples for $n = 8$.

Table 10.4: Examples of forgeries for HERN

Input data	**associated data-only**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	D8A4ADC965EECE56330E5CC01A53C928	D8A4ADC965EECE56330E5CC01A53C928
AD	CA5F	CA5F00
M	Empty string	Empty string
CT	Empty string	Empty string
T	00FC40BF26954B37993E9C56C6C49ACA	FC40BF26954B37993E9C56C6C49ACAB6
Input data	**ciphertext-only**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	3E1327BCC61246AC87901E0922C1A354	3E1327BCC61246AC87901E0922C1A354
AD	9524	9524
M	8500	85
CT	0D00	0D
T	8472B9D92F6AAC22CE3F188CC13D711C	008472B9D92F6AAC22CE3F188CC13D71
Input data	**associated data and ciphertext**	
K	000102030405060708090A0B0C0D0E0F	000102030405060708090A0B0C0D0E0F
N	7B8A185D3B33E4F906E02F291BEF6C06	7B8A185D3B33E4F906E02F291BEF6C06
AD	4328	432800
M	00	Empty string
CT	00	Empty string
T	A72C78D89FAD7A7D785EF13AB2EC085B	A72C78D89FAD7A7D785EF13AB2EC085B

10.6 Summary

In this chapter, we have demonstrated a series of practical forgery attacks on Limdolen and HERN which defeat the designers' claim of 128(256) and 128-bit integrity security of Limdolen-128(256) and HERN, respectively. For both variants of Limdolen, we have shown the constructions of forgeries which require a single encryption and a single decryption query, and have a success probability of 1. For HERN we have presented round independent associated data-only, ciphertext-only and associated data and ciphertext forgeries which have the success rate of 1 after 2(2), 4(2) and 2(1) encryption(decryption) queries, respectively. Following our attack, both submissions were eliminated from round 2 of the NIST LWC project.

Possible fixes. To resist our attacks on Limdolen, the period 2 masking sequence has to be replaced by a sequence with unpredictable properties. A simple fix for HERN seems to be to complement a state bit (except the last bit of each register) after $640 + |AD|$ and $640 + |AD| + |M|$ clock cycles. However, the security needs to be studied thoroughly.

Chapter 11

Conclusions and Future Work

In this thesis, we have presented novel research contributions in the area of lightweight cryptography, including both the design and cryptanalysis. We have proposed lightweight permutations, and instantiated them in two different modes to provide several AEAD and hash algorithms with varying security levels. We have also analyzed existing lightweight ciphers with respect to cube attacks, correlated sequences and forgery attacks. In the following, we give the concluding remarks and then discuss the potential future research directions.

11.1 Concluding Remarks

The contributions of the thesis consists of three parts. Part I is composed of Chapters 3-5, which present new lightweight cryptographic permutations. Part II contains Chapters 6 and 7, which discuss how to design modes in the sponge framework and their instantiations using these lightweight cryptographic permutations, and Part III includes Chapter 8-10, which provide cryptanalysis results for lightweight ciphers.

In Chapter 3, we have first introduced the design of sLiSCP permutation, a lightweight permutation which consists of hardware friendly bitwise XOR and AND operations. We adopted a combination of large Simeck sboxes (with sizes 48 and 64 bits) and type-II generalized Feistel round (with 4 branches) to design its step function. We then analyzed the cryptographic properties such as differential, linear and algebraic degree of Simeck sboxes, and thus sLiSCP using SAT/SMT and MILP tools. Later we noticed that the 2 Simeck sboxes which are placed in between the odd and even branches can in fact be positioned at odd branches only. This saved the cost of two extra 48(64)-bit registers and lead to the design of sLiSCP-light, which has lower area and higher throughput than sLiSCP. Our security analysis later revealed that sLiSCP-light has better algebraic properties than sLiSCP, but it is weaker in differential and linear properties, a trade-off one would expect while optimizing design parameters.

The former two permutations cannot be used to achieve hash functionality with a 256-bit message digest and 128-bit collision security. Motivated by this requirement, in Chapter 4, we generalized the structures of sLiSCP and sLiSCP-light to five branches, and proposed a 320-bit permutation ACE. Although ACE utilizes 3 Simeck sboxes each of size 64 bits, we found that there exist linear layers which can offer better security properties than the traditional left blockwise shuffle. Accordingly, we chose $(3, 2, 0, 4, 1)$ as the linear layer of ACE and then analyzed the security of the permutation with respect to distinguishing attacks. In Chapter 5, we extended our design approach to cryptographic permutations which are indistinguishable from random permutations, and at the same time can guarantee certain theoretical randomness properties. As a result, we proposed WAGE, a permutation defined over the extension field $\mathbb{F}_{2^7}$, which can be transformed to the original WG stream cipher with simple tweaks.

In Chapter 6, we discussed the need for uniform circuitry for achieving multiple cryptographic functionalities using a cryptographic permutation in a sponge mode, and consequently, introduced the idea of the unified round function. In terms of uniformity and the number of domain separator bits (2 in our case), this is the minimum one can achieve. We have presented AEAD and hash schemes with varying data limits and security levels. One of our interesting proposal is Hash-[ACE] and AE-[ACE] for which key size = nonce size = tag size = number of rounds = security level = 128, and rate = 64. Other AEAD instances with a 64-bit rate provide 128-bit security, but in hash mode the collision security is limited to 96 bits with the same rate. In Chapter 7, we presented Spoc which offers 112-bit security (data limit 2^{50} bytes) when instantiated with sLiSCP-light-192 and 64-bit rate. The same security could not be achieved with the traditional sponge AEAD mode. However, the trade-off is that one now needs an additional 64-bit XOR, 64-bit multiplexers and 4 domain separator bits. In a nutshell, our proposed schemes in addition to the 4 NIST LWC round 2 candidates, namely ACE, Spix, Spoc and WAGE have different performance and hence target a wide range of applications.

In Chapter 8, we have used the division property based cube attacks to analyze the nonlinear initialization phase of the lightweight stream cipher WG-5. In our analyis, we modeled the divison property of each component of the cipher as a set of linear inequalites. The reduced-round WG-5 is then translated to an optimization model whose solutions are the secret key bits involved in the superpoly for a given cube. We used multiple cubes and then presented a key recovery attack on 24 (out of 64) rounds with data and time complexity of $2^{6.32}$ and $2^{76.81}$, respectively. We further provided an argument to show that the WG-5 design parameters in terms of feedback and tap positions are more resistant to cube attacks than Grain-128a and Trivium. We expect that the analysis on WG-5 can provide more confidence in the design (especially number of rounds) of WAGE.

In Chapter 9, we have proposed a novel property of block ciphers called correlated sequences which could extend the number of rounds of classic meet-in-the-middle attacks. As an application, we have shown the construction of length 8 correlated sequences of Simon-32/64 and Simeck-32/64, and utilized them for MitM attacks on both ciphers which covered 27 (out of 32) rounds. The attack requires 3 known plaintext-ciphertext pairs and has a time complexity close to that of 2^{63} 27-round encryptions. The successful probability of our attack is 1. It is worth noting that the sLiSCP, sLiSCP-light and ACE permutations are based on reduced-round Simeck. Here we emphasize that correlated sequences are not applicable to them as the permutations are public.

Finally, in Chapter 10, we have presented simple, yet practical and devastating, forgeries on two NIST LWC round 1 candidates Limdolen and HERN. For Limdolen, we observed that its masking values have a period of 2. We exploited this observation and constructed forgeries by adding, removing or permutating an arbitrary number of blocks. For HERN, we have found that associated data and message processing phases are not distinguishable. Consequently, we have shown the construction of forgeries by appending or removing a sequence of zero bits in associated data or message. We further discussed that fixes are simple but require design changes (especially for Limdolen).

11.2 Future Research Problems

Lightweight alternatives of Simeck sboxes. An sbox with a smaller area and similar cryptographic properties as of the Simeck sbox can reduce the area of sLiSCP, sLiSCP-light and ACE permutations. Thus, it is worth exploring different design options at the sbox level. For instance,

- Use multiple 4-bit sboxes and combine them with a cheap linear layer, e.g., lightweight MDS matrix or a variant of reduced-round PRESENT [42] or GIFT [21].
- Simeck-like round function with equal number of ANDs and XORs (as AND is cheaper in hardware than XOR).

Optimal linear layers for GFS type-II structures. While designing ACE we found that there exist linear layers which can offer better differential and linear properties than left blockwise shuffle. A few interesting problems for designers in this direction are as follows.

1. Let $n \not\equiv 0 \bmod 2$ and $n \geq 6$ be the number of branches of GFS type-II structure and r denote the number of rounds. Find a permutation π of $(0, 1, \ldots, n-1)$ or a class of permutations which gives the maximum of minimum number of active sboxes for $r = 2n, 3n$ and $4n$.

2. If there exists a class of permutations in (1), then how are the permutations in this class related ? Is it possible to generalize this class for any n?

Note that there exist results in the literature which targeted optimal linear layers based on diffusion [55, 46, 124]. However, the differential property is much stronger than diffusion and we are not aware of any such results.

Further analysis of WAGE. The WAGE permutation requires further security analysis as this is a completely new design. We could provide tighter bounds for the maximum differential characteristic probability by finding the minimum number of active sboxes for at least 75 rounds.

Applications of correlated sequences. The current attacks on Simon-32/64 and Simeck-32/64 cover 27 rounds by utilizing length 8 correlated sequences. Thus, it is natural to ask how to improve it further. We believe that improvement in number of rounds and time complexity can be achieved by finding correlated sequences of length at least 9. Furthermore, such sequences may have similar applications to other variants of Simon and Simeck. In addition, investigating the underlying ciphers' structure to construct correlated sequences is another interesting problem.

Masking schemes of NIST LWC round 2 candidates. Most of the NIST LWC round 2 candidates (especially the ones based on block ciphers) use varying masking schemes to randomize the input of primitives. Thus, it would be interesting to look at their indistinguishable properties or period of secret masks, and whether forgery attacks are possible in those schemes.

Bibliography

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

[2] Gurobi: MILP optimizer. http://www.gurobi.com/

[3] SageMath. http://www.sagemath.org/

[4] eSTREAM: The ECRYPT stream cipher project. https://www.ecrypt.eu.org/stream/

[5] NIST Lightweight Cryptography Standardization Project. https://csrc.nist.gov/projects/lightweight-cryptography, (accessed 20 Nov 2019).

[6] EPC radio frequency identity protocols class-1 generation-2 UHF RFID, protocol for communications at 860 MHz - 960 MHz version 2. EPCglobal Inc. specification documents, Apr. 2015. https://www.gs1.org/sites/default/files/docs/epc/Gen2_Protocol_Standard.pdf.

[7] Aagaard, M., AlTawy, R., Gong, G., Mandal, K., and Rohit, R. ACE: An authenticated encryption and hash algorithm. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[8] Aagaard, M., AlTawy, R., Gong, G., Mandal, K., Rohit, R., and Zidaric, N. WAGE: An authenticated cipher. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[9] Aagaard, M. D., Gong, G., and Mota, R. K. Hardware implementations of the WG-5 cipher for passive RFID tags. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013 , pp. 29-34. (2013)

[10] Adams, C. The CAST-256 encryption algorithm, RFC2612. https://tools.ietf.org/html/rfc2612

[11] Agren, M., Hell, M., Johansson, T., and Meier, W. Grain-128a: A new version of Grain-128 with optional authentication. *Int. J. Wire. Mob. Comput.*, vol. 5, no. 1 (Dec. 2011), pp. 48-59. (2011)

[12] AlTawy, R., Gong, G., He, M., Jha, A., Mandal, K., Nandi, M., and Rohit, R. SpoC: An authenticated cipher. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[13] AlTawy, R., Gong, G., He, M., Mandal, K., and Rohit, R. SPIX: An authenticated cipher. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[14] AlTawy, R., Rohit, R., He, M., Mandal, K., Yang, G., and Gong, G. sLiSCP: Simeck-based permutations for lightweight sponge cryptographic primitives. In: Adams C., Camenisch J. (eds), *SAC 2017. LNCS*, vol. 10719, pp. 129-150. Springer, Cham (2017)

[15] AlTawy, R., Rohit, R., He, M., Mandal, K., Yang, G., and Gong, G. sLiSCP-light: Towards hardware optimized sponge-specific cryptographic permutations. *ACM Transactions on Embedded Computing Systems*, vol. 17, no. 4, article no. 81, pp. 1-26, 2018.

[16] AlTawy, R., Rohit, R., He, M., Mandal, K., Yang, G., and Gong, G. Towards a cryptographic minimal design: The sLiSCP family of permutations. *IEEE Transactions on Computers*, vol. 67, no. 9, pp. 1341-1358, 2018.

[17] Aumasson, J.-P., Henzen, L., Meier, W., and Naya-Plasencia, M. Quark: A lightweight hash. *Journal of Cryptology*, vol. 26, no. 2, pp. 313-339, 2013.

[18] Aumasson, J.-P., Jovanovic, P., and Neves, S. NORX: Parallel and scalable aead. In: M. Kutyłowski and J. Vaidya (eds), *ESORICS 2014. LNCS*, vol. 8713, pp. 19-36. Springer, Cham (2014)

[19] Aumasson, J.-P., Jovanovic, P., and Neves, S. NORX8 and NORX16: Authenticated encryption for low-end systems. Cryptology ePrint Archive, Report 2015/1154, 2015. http://eprint.iacr.org/2015/1154

[20] Babbage, S., and Dodd, M. The MICKEY stream ciphers. In: *New stream cipher designs. LNCS*, vol. 4986, pp. 191-209. Springer, Heidelberg (2008)

[21] Banik, S., Pandey, S. K., Peyrin, T., Sasaki, Y., Sim, S. M., and Todo, Y. GIFT: A small present. In: Fischer W., Homma N. (eds), *CHES 2017. LNCS*, vol. 10529, pp. 321-345. Springer, Cham (2017)

[22] Bao, Z., Chakraborti, A., Datta, N., Guo, J., Nandi, M., Peyrin, T., and Yasuda, K. Photon-Beetle: Authenticated encryption and hash family. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[23] Beaulieu, R., Treatman-Clark, S., Shors, D., Weeks, B., Smith, J., and Wingers, L. The Simon and Speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.

[24] Beierle, C., Biryukov, A., dos Santos, L. C., Großschädl, J., Perrin, L., Udovenko, A., Velichkov, V., Wang, Q., and Biryukov, A. SCHWAEMM and ESCH: Lightweight authenticated encryption and hashing using the SPARKLE permutation family. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[25] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., and Sim, S. M. The SKINNY family of block ciphers and its low-latency variant MANTIS. In: M. Robshaw and J. Katz (eds), *CRYPTO 2016. LNCS*, vol. 9815, pp. 123-153. Springer, Heidelberg (2016)

[26] Beierle, C., Leander, G., Moradi, A., and Rasoolzadeh, S. CRAFT: Lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Transactions on Symmetric Cryptology* 2019(1), pp. 5-45, 2019.

[27] Bellare, M., Canetti, R., and Krawczyk, H. Keying hash functions for message authentication. In: Koblitz N (eds), *CRYPTO 1996. LNCS*, vol. 1109, pp. 1-15. Springer, Heidelberg (1996)

[28] Bellizia, D., Berti, F., Bronchain, O., Cassiers, G., Duval, S., Guo, C., Leander, G., Leurent, G., Levi, I., Momin, C., et al. Spook: Sponge-based leakage-resilient authenticated encryption with amasked tweakable block cipher. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[29] Bernstein, D. J., Kölbl, S., Lucks, S., Massolino, P. M. C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.-X., Todo, Y., and Viguier, B. Gimli: A cross-platform permutation. In: Fischer W., Homma N. (eds), *CHES 2017. LNCS*, vol. 10529, pp. 299–320. Springer, Cham (2017)

[30] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. CAEASR submission: Ketje v2, 2014. http://ketje.noekeon.org/Ketjev2-doc2.0.pdf.

[31] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Sponge functions. In: *ECRYPT hash workshop, 2007.* http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.8103&rep=rep1&type=pdf

[32] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Keccak specifications. *Submission to NIST (Round 2, SHA3 competition), 2009.*

[33] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Sponge-based pseudo-random number generators. In: S. Mangard and F.-X. Standaert (eds), *CHES 2010*. *LNCS*, vol. 6225, pp. 33–47. Springer, Heidelberg (2010)

[34] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. On the security of the keyed sponge construction. In: *Symmetric Key Encryption Workshop 2011.*

[35] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Duplexing the sponge: Single-pass authenticated encryption and other applications. In: A. Miri and S. Vaudenay (eds), *SAC 2012*. *LNCS*, vol. 7118, pp. 320-337. Springer, Heidelberg (2012)

[36] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Permutation-based encryption, authentication and authenticated encryption. In: *DIAC 2012.*

[37] Bertoni, G., Daemen, J., Peeters, M., and Van Assche, G. Cryptographic sponge functions, 2014. http://sponge.noekeon.org/CSF-0.1.pdf

[38] Biham, E., and Shamir, A. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, vol. 4, no. 1, pp. 3-72, 1991.

[39] Black, J., and Rogaway, P. A block-cipher mode of operation for parallelizable message authentication. In: Knudsen L.R. (eds), *EUROCRYPT 2002*. *LNCS*, vol. 2332, pp. 384–397. Springer, Heidelberg (2002)

[40] Blondeau, C., Bogdanov, A., and Wang, M. In: I. Boureanu, P. Owesarski, and S. Vaudenay (eds), *ACNS 2014*. *LNCS*, vol. 8479, pp. 271-288. Springer, Cham (2014)

[41] Bogdanov, A., Knežević, M., Leander, G., Toz, D., Varıcı, K., and Verbauwhede, I. Spongent: A lightweight hash function. In: B. Preneel and T. Takagi (eds), *CHES 2011*. *LNCS*, vol. 6917, pp. 312–325. Springer, Heidelberg (2011)

[42] Bogdanov, A., Knudsen, L. R., Leander, G., Paar, C., Poschmann, A., Robshaw, M. J. B., Seurin, Y., and Vikkelsoe, C. PRESENT: An ultra-lightweight block cipher. In: P. Paillier and I. Verbauwhede (eds), *CHES 2007*. *LNCS*, vol. 4727, pp. 450–466. Springer, Heidelberg (2007)

[43] Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E. B., Knezevic, M., Knudsen, L. R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S. S., and Yalçın, T. PRINCE: A low-latency block cipher for pervasive computing applications. In: Wang X., Sako K. (eds), *ASIACRYPT 2012. LNCS*, vol. 7658, pp. 208–225. Springer, Heidelberg (2012)

[44] Burwick, C., Coppersmith, D., DAvignon, E., Gennaro, R., Halevi, S., Jutla, C., Matyas Jr, S. M., OConnor, L., Peyravian, M., Safford, D., et al. MARS: A candidate cipher for AES. NIST AES Proposal, 1998.

[45] Canteaut, A. Differential cryptanalysis of Feistel ciphers and differentially uniform mappings. In: Adams C., Just M. (eds), *SAC 1997*, pp. 172–184.

[46] Cauchois, V., Gomez, C., and Thomas, G. General diffusion analysis: How to find optimal permutations for generalized type-II feistel schemes. *IACR Transactions on Symmetric Cryptology* 2019(1), pp. 264-301, 2019.

[47] Chen, H., and Wang, X. Improved linear hull attack on round-reduced simon with dynamic key-guessing techniques. In: Peyrin T. (eds), *FSE 2016. LNCS*, vol. 9783, pp. 428–449. Springer, Heidelberg (2016)

[48] Chen, L., and Gong, G. Communication system security. *Chapman and Hall/CRC*, 2012.

[49] Chu, Z., Chen, H., Wang, X., Dong, X., and Li, L. Improved integral attacks on simon32 and simon48 with dynamic key-guessing techniques. *Security and Communication Networks*, 2018. https://doi.org/10.1155/2018/5160237.

[50] Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., and Van Keer, R. Xoodyak: A lightweight cryptographic scheme. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[51] Daemen, J., Massolino, P. M. C., and Rotella, Y. The Subterranean 2.0 cipher suite. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[52] Daemen, J., and Rijmen, V. The block cipher Rijndael. In: Quisquater JJ., Schneier B. (eds), *CARDIS 1998. LNCS*, vol. 1820, pp. 277–284. Springer, Heidelberg (1998)

[53] De Cannière, C. Trivium: A stream cipher construction inspired by block cipher design principles. In: Katsikas S.K., Lopez J., Backes M., Gritzalis S., Preneel B. (eds), *ISC 2006. LNCS*, vol. 4176, pp. 171-186. Springer, Heidelberg (2006)

[54] Derbez, P., and Fouque, P.-A. Automatic search of meet-in-the-middle and impossible differential attacks. In: Robshaw M., Katz J. (eds), *CRYPTO 2016. LNCS*, vol. 9815, pp. 157–184. Springer, Heidelberg (2016)

[55] Derbez, P., Fouque, P.-A., Lambin, B., and Mollimard, V. Efficient search for optimal diffusion layers of generalized feistel networks. *IACR Transactions on Symmetric Cryptology* 2019(2), pp. 218-240, 2019.

[56] Diffie, W., and Hellman, M. New directions in cryptography. *IEEE Transactions on Information Theory*, vol 22, no. 6, pp. 644-654, 1976.

[57] Diffie, W., and Hellman, M. E. Special feature exhaustive cryptanalysis of the nbs data encryption standard. *Computer*, vol. 10, no. 6, pp. 74–84, 1977.

[58] Dinu, D., Perrin, L., Udovenko, A., Velichkov, V., Großschädl, J., and Biryukov, A. Design strategies for arx with provable bounds: Sparx and lax. In: J. H. Cheon and T. Takagi (eds), *ASIACRYPT 2016. LNCS*, vol. 10031, pp. 484–513. Springer, Heidelberg (2016)

[59] Dinur, I., and Shamir, A. Cube attacks on tweakable blackbox polynomials. In: Joux A. (eds), *EUROCRYPT 2009. LNCS*, vol. 5479, pp. 278–299. Springer, Heidelberg (2009)

[60] Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M. Ascon v1.2. Submission to the CAESAR competition: http://competitions.cr.yp.to/round3/asconv12.pdf. Submission to *NIST Lightweight Cryptography Standardization Project* (announced as round 2 candidate on August 30, 2019).

[61] Eastlake, D., and Jones, P. US secure hash algorithm 1 (SHA1), RFC3174, 2001.

[62] El-Razouk, H., Reyhani-Masoleh, A., and Gong, G. New hardware implementations of WG(29,11) and WG-16 stream ciphers using polynomial basis. *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 2020-2035, 2015.

[63] Fan, X., Mandal, K., and Gong, G. WG-8: A lightweight stream cipher for resource-constrained smart devices. In: K. Singh and A. K. Awasthi (eds), *Quality, Reliability, Security and Robustness in Heterogeneous Networks 2013.* LNICST, vol. 115, pp. 617–632. Springer, Heidelberg (2013)

[64] Fan, X., Zidaric, N., Aagaard, M., and Gong, G. Efficient hardware implementation of the stream cipher WG-16 with composite field arithmetic. In *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices, TrustED '13. ACM*, pp. 21-34.

[65] Feistel, H. Block cipher cryptographic system. US Patent 3798359A, 1974.

[66] Fu, K., Sun, L., and Wang, M. New integral attacks on simon. *IET Information Security*, vol. 11, no. 5, pp. 277-286, 2016.

[67] Golomb, S. W., et al. Shift register sequences. *Aegean Park Press*, 1967.

[68] Golomb, S. W., and Gong, G. Signal design for good correlation: for wireless communication, cryptography, and radar. *Cambridge University Press*, 2005.

[69] Gueron, S., and Mouha, N. Simpira v2: A family of efficient permutations using the AES round function. In: J. H. Cheon and T. Takagi (eds), *ASIACRYPT 2016. LNCS*, vol. 10031, pp. 95–125. Springer, Heidelberg (2016)

[70] Guo, J., Peyrin, T., and Poschmann, A. The Photon family of lightweight hash functions. In: P. Rogaway (eds), *CRYPTO 2011. LNCS*, vol. 6841, pp. 222–239. Springer, Heidelberg (2011)

[71] Guo, J., Peyrin, T., Poschmann, A., and Robshaw, M. The LED block cipher. In: B. Preneel and T. Takagi (eds), *CHES 2011. LNCS*, vol. 6917, pp. 326–341. Springer, Heidelberg (2011)

[72] Hamann, M., Krause, M., and Meier, W. Lizard: A lightweight stream cipher for power-constrained devices. *IACR Transactions on Symmetric Cryptology* 2017(1), pp. 45-79. (2017)

[73] Hell, M., Johansson, T., Maximov, A., and Meier, W. A stream cipher proposal: Grain-128. *IEEE International Symposium on Information Theory*, pp. 1614-1618, 2006

[74] Hong, D., Sung, J., Hong, S., Lim, J., Lee, S., Koo, B.-S., Lee, C., Chang, D., Lee, J., Jeong, K., Kim, H., Kim, J., and Chee, S. Hight: A new block cipher suitable for low-resource device. In: Goubin L., Matsui M (eds), *CHES 2006. LNCS*, vol. 4249, pp. 46–59. Springer, Heidelberg (2006)

[75] Hsu, C.-L., and Lin, J. C.-C. An empirical examination of consumer adoption of internet of things services: Network externalities and concern for information privacy perspectives. *Computers in Human Behavior*, vol. 62, pp. 516–527, 2016.

[76] Hu, H., and Gong, G. Periods on two kinds of nonlinear feedback shift registers with time varying feedback functions. *International Journal of Foundations of Computer Science*, vol.22, no. 6, pp. 1317-1329, 2011.

[77] Iwata, T., and Kurosawa, K. OMAC: One-Key CBC MAC. In: Johansson T. (eds), *FSE 2003. LNCS*, vol. 2887, pp. 129–153. Springer, Heidelberg (2003)

[78] Jean, J., Nikolic, I., Peyrin, T., and Seurin, Y. Deoxys v1. 41. Submitted to CAESAR Competition. https://competitions.cr.yp.to/round3/deoxysv14.pdf

[79] Jovanovic, P., Luykx, A., and Mennink, B. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In: P. Sarkar and T. Iwata (eds), *ASIACRYPT 2014*. *LNCS*, vol. 8873, pp. 85–104. Springer, Heidelberg (2014)

[80] Juels, A., and Weis, S. A. Authenticating pervasive devices with human protocols. In: Shoup V. (eds), *CRYPTO 2005*. *LNCS*, vol. 3621, pp. 293–308. Springer, Heidelberg (2005)

[81] Kahn, D. The Codebreakers: The comprehensive history of secret communication from ancient times to the internet. *Simon and Schuster*, 1996.

[82] Katz, J., Menezes, A. J., Van Oorschot, P. C., and Vanstone, S. A. Handbook of applied cryptography. *CRC press*, 1996.

[83] Keliher, L. Exact maximum expected differential and linear probability for two-round advanced encryption standard. *IET Information Security*, vol. 1, no. 2, pp. 53–57, 2007.

[84] Knežević, M., Nikov, V., and Rombouts, P. Low-latency encryption - Is lightweight = light+ wait? In: Prouff E., Schaumont P. (eds), *CHES 2012*. *LNCS*, vol. 7428, pp. 426–446. Springer, Heidelberg (2012)

[85] Knudsen, L., and Wagner, D. Integral cryptanalysis. In: Daemen J., Rijmen V. (eds), *FSE 2012*. *LNCS*, vol. 2365, pp. 112–127. Springer, Heidelberg (2012)

[86] Kölbl, S., Leander, G., and Tiessen, T. Observations on the Simon block cipher family. In: R. Gennaro and M. Robshaw (eds), *CRYPTO 2015*. *LNCS*, vol. 9215, pp. 258–269. Springer, Heidelberg (2015)

[87] Kölbl, S., and Roy, A. A brief comparison of Simon and Simeck. In: Bogdanov A. (eds), *LightSec 2016*. *LNCS*, vol. 10098, pp. 69–88. Springer, Cham (2016)

[88] Krovetz, T., and Rogaway, P. The software performance of authenticated-encryption modes. In: Joux A. (eds), *FSE 2011*. *LNCS*, vol. 6733, pp. 306–327. Springer, Heidelberg (2011)

[89] Lai, X. Higher order derivatives and differential cryptanalysis. In: Blahut R.E., Costello D.J., Maurer U., Mittelholzer T. (eds), *Communications and Cryptography 1994*. *LNCS*, vol. 276, pp. 227–233. Springer, MA (1994)

[90] Lai, X., Massey, J. L., and Murphy, S. Markov ciphers and differential cryptanalysis. In: Davies D.W. (eds), *EUROCRYPT 1991. LNCS*, vol. 547, pp. 17–38. Springer, Heidelberg (1991)

[91] Liskov, M., Rivest, R. L., and Wagner, D. Tweakable block ciphers. *Journal of Cryptology*, vol. 24, no. 3, pp. 588-613, 2011.

[92] Liu, Y., Sasaki, Y., Song, L., and Wang, G. Cryptanalysis of reduced sLiSCP permutation in sponge-hash and duplex-AE modes. In: C. Cid and J. Michael J. Jacobson (eds), *SAC 2018. LNCS*, vol. 11349, pp. 92–114. Springer, Cham (2018)

[93] Luo, Y., Chai, Q., Gong, G., and Lai, X. A lightweight stream cipher WG-7 for RFID encryption and authentication. In: *IEEE Global Telecommunications Conference GLOBECOM 2010*, pp. 1-6.

[94] Mandal, K., Gong, G., Fan, X., and Aagaard, M. Optimal parameters for the WG stream cipher family. *Cryptography and Communications*, vol. 6, no. 2, pp. 117-135, 2014.

[95] Matsui, M., and Yamagishi, A. A new method for known plaintext attack of FEAL cipher. In: Rueppel R.A. (eds), *EUROCRYPT 1992. LNCS*, vol. 658, pp. 81–91. Springer, Heidelberg (1992)

[96] McGrew, D. A., and Viega, J. The security and performance of the galois/counter mode (GCM) of operation. In: Canteaut A., Viswanathan K. (eds), *INDOCRYPT 2004. LNCS*, vol. 3348, pp. 343–355. Springer, Heidelberg (2004)

[97] McKay, K., Bassham, L., Sönmez Turan, M., and Mouha, N. Report on lightweight cryptography (NISTIR8114), 2017.

[98] Meher, C. E. Limdolen: A lightweight authenticated encryption algorithm. NIST LWC round 1 submssion. https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/Limdolen-Spec.pdf.

[99] Mikhalev, V., Armknecht, F., and Mller, C. On ciphers that continuously access the non-volatile key. *IACR Transactions on Symmetric Cryptology* 2016(2), pp. 218-240, 2016.

[100] Nawaz, Y., and Gong, G. The WG stream cipher. ECRYPT Stream Cipher Project Report 2005.

[101] Nawaz, Y., and Gong, G. Wg: A family of stream ciphers with designed randomness properties. *Information Sciences*, vol. 178, no. 7, pp. 1903-1916, 2008.

[102] NIST Advanced Encryption Standard. Federal Information Processing Standards Publication 197, 2001. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf.

[103] NIST Secure Hash Standard. Federal Information Processing Standards Publication 180-4, 2015. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

[104] NIST SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication 202, 2015. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.

[105] Nyberg, K. Linear approximation of block ciphers. In: De Santis A. (eds), *EUROCRYPT 1994*. *LNCS*, vol. 950, pp. 439–114. Springer, Heidelberg (1994).

[106] Nyberg, K. Generalized feistel networks. In: K. Kim and T. Matsumoto (eds), *ASIACRYPT 1996*. *LNCS*, vol. 1163, pp. 91–104. Springer, Heidelberg (1996)

[107] Nyberg, K., and Knudsen, L. R. Provable security against a differential attack. *Journal of Cryptology*, vol. 8, no. 1, pp. 27–37, 19951 (19

[108] Paar, C., and Pelzl, J. Understanding cryptography: A textbook for students and practitioners. *Springer Science & Business Media*, 2009.

[109] Qiao, K., Hu, L., and Sun, S. Differential analysis on Simeck and Simon with dynamic key-guessing techniques. In: Camp O., Furnell S., Mori P. (eds), *ICISSP 2016*. *LNCS*, vol. 691, pp. 64–85. Springer, Cham (2016)

[110] Qin, L., Chen, H., and Wang, X. Linear hull attack on round-reduced simeck with dynamic key-guessing techniques. In: Liu J., Steinfeld R. (eds) *Information Security and Privacy, ACISP 2016*. *LNCS*, vol. 9723, pp. 409–424. Springer, Cham (2016)

[111] Rivest, R. The MD5 message-digest algorithm (RFC1321). https://tools.ietf.org/html/rfc1321

[112] Rivest, R. L., Robshaw, M. J., Sidney, R., and Yin, Y. L. The RC6 block cipher. https://people.csail.mit.edu/rivest/pubs/RRSY98.pdf

[113] Rogaway, P. Authenticated-encryption with associated-data. In: *CCS 2002*, *ACM*, pp. 98–107.

[114] Rogaway, P., Bellare, M., and Black, J. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security*, vol. 6, no. 3, pp. 365–403, 2003.

[115] Rohit, R., AlTawy, R., and Gong, G. Milp-based cube attack on the reduced-round WG-5 lightweight stream cipher. In: O'Neill M. (eds), *IMACC 2017. LNCS*, vol. 10655, pp. 333–351. Springer, Cham (2017)

[116] Rohit, R., and Gong, G. Meet-in-the-middle attack using correlated sequences and its applications to Simon-like ciphers. http://cacr.uwaterloo.ca/techreports/2018/cacr2018-07.pdf.

[117] Rohit, R., and Gong, G. Practical forgery attacks on limdolen and Hern. *NIST Lightweight Cryptography Workshop, 2019.* Also available at Cryptology ePrint Archive, Report 2019/907, 2019. http://eprint.iacr.org/2019/907

[118] Rønjom, S. Improving algebraic attacks on stream ciphers based on linear feedback shift register over $\mathbb{F}_{2^k}$. *Designs, Codes and Cryptography*, vol. 82, no. 1-2, pp. 27–41, 017.

[119] Shirai, T., Shibutani, K., Akishita, T., Moriai, S., and Iwata, T. The 128-bit blockcipher CLEFIA. In: Biryukov A. (eds), *FSE 2007. LNCS*, vol. 4593, pp. 181-195. Springer, Heidelberg (2007)

[120] Song, L., Hu, L., Ma, B., and Shi, D. Match box meet-in-the-middle attacks on the Simon family of block ciphers. In: Eisenbarth T., Öztürk E. (eds), *LightSec 2014. LNCS*, vol. 8898, pp. 140–151. Springer, Cham (2014)

[121] Standard, D. E. FIPS publication 46, 1977.

[122] Kölbl, S. CryptoSMT: An easy to use tool for cryptanalysis of symmetric primitives. https://github.com/kste/cryptosmt.

[123] Sun, L., Fu, K., and Wang, M. Improved zero-correlation cryptanalysis on simon. In: Lin D., Wang X., Yung M. (eds), *Inscrypt 2015. LNCS*, vol. 9589, pp. 125–143. Springer, Cham (2014)

[124] Suzaki, T., and Minematsu, K. Improving the generalized feistel. In: Hong S., Iwata T. (eds), *FSE 2010. LNCS*, vol. 6147, pp. 19–39. Springer, Heidelberg (2010)

[125] Suzaki, T., Minematsu, K., Morioka, S., and Kobayashi, E. Twine: A lightweight block cipher for multiple platforms. In: Knudsen L.R., Wu H. (eds), *SAC 2012. LNCS*, vol. 7707, pp. 339–354. Springer, Heidelberg (2012)

[126] Todo, Y. Structural evaluation by generalized integral property. In: E. Oswald and M. Fischlin (eds), *EUROCRYPT 2015. LNCS*, vol. 9056, pp. 287–314. Springer, Heidelberg (2015)

[127] Todo, Y., Isobe, T., Hao, Y., and Meier, W. Cube attacks on non-blackbox polynomials based on division property. In: Katz J., Shacham H. (eds), *CRYPTO 2017. LNCS*, vol. 10403, pp. 250–279. Springer, Cham (2017)

[128] Todo, Y., and Morii, M. Bit-based division property and application to Simon family. In: Peyrin T. (eds), *FSE 2016. LNCS*, vol. 9783, pp. 357–377. Springer, Heidelberg (2016)

[129] Vielhaber, M. Breaking One.Fivium by AIDAaan algebraic iv differential attack. Cryptology ePrint Archive, Report 2007/413, 2007. https://eprint.iacr.org/2007/413.pdf

[130] Wang, N., Wang, X., Jia, K., and Zhao, J. Differential attacks on reduced simon versions with dynamic key-guessing techniques. Cryptology ePrint Archive, Report 2014/448, 2014. https://eprint.iacr.org/2014/448.pdf.

[131] Wang, Q., Liu, Z., Varıcı, K., Sasaki, Y., Rijmen, V., and Todo, Y. Cryptanalysis of reduced-round simon32 and simon48. In: Meier W., Mukhopadhyay D. (eds), *Indocrypt 2014. LNCS*, vol. 8885, pp. 143–160. Springer, Cham (2014)

[132] Wheeler, D., and Needham, R. TEA: A tiny encryption algorithm. In: Preneel B. (eds), *FSE 1994. LNCS*, vol. 1008, pp. 97–110. Springer, Heidelberg (2005)

[133] Wu, H. ACORN: a lightweight authenticated cipher (v3). https://competitions.cr.yp.to/round3/acornv3.pdf.

[134] Xiang, Z., Zhang, W., Bao, Z., and Lin, D. Applying milp method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: J. H. Cheon and T. Takagi (eds), *ASIACRYPT 2016. LNCS*, vol. 10031, pp. 648–678. Springer, Heidelberg (2016)

[135] Yang, G., Zhu, B., Suder, V., Aagaard, M. D., and Gong, G. The Simeck family of lightweight block ciphers. In: T. Güneysu and H. Handschuh (eds), *CHES 2015. LNCS*, vol. 9293, pp. 307–329. Springer, Heidelberg (2015)

[136] Yap, H., Khoo, K., Poschmann, A., and Henricksen, M. EPCBC: A block cipher suitable for electronic product code encryption. In: Lin D., Tsudik G., Wang X.(eds), *CANS 2011. LNCS*, vol. 7092, pp. 76–97. Springer, Heidelberg (2011)

[137] Ye, D., Shi, D., Ma, Y., and Wang, P. HERN and HERON: Lightweight AEAD and hash constructions based on thin sponge (v1). https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/.

[138] Zhang, K., Guan, J., Hu, B., and Lin, D. Integral cryptanalysis on Simeck. *ICIST 2016. IEEE*, pp. 216–222. (2016)

[139] Zhang, K., Guan, J., Hu, B., and Lin, D. Security evaluation on Simeck against zero-correlation linear cryptanalysis. *IET Information Security*, vol 12, no. 1, pp. 87-93, 2018.

[140] Zheng, Y., Matsumoto, T., and Imai, H. On the construction of block ciphers provably secure and not relying on any unproved hypotheses. In: Brassard G. (eds), *CRYPTO 1989*. LNCS, vol. 435, pp. 461–480. Springer, Heidelberg (1989)

[141] Zhu, B., and Gong, G. Multidimensional meet-in-the-middle attack and its applications to Katan32/48/64. *Cryptography and Communications*, vol. 6, no. 4, pp. 313–333, 2014.

[142] Zidaric, N., Aagaard, M., and Gong, G. Hardware optimizations and analysis for the WG-16 cipher with tower field arithmetic. *IEEE Transactions on Computers*, vol. 68, no. 1, pp. 67-82, 2019.

APPENDICES

Appendix A

Appendices: WAGE

A.1 Round Constants of WAGE

Round i	**Round constant** (rc_1^i, rc_0^i)									
0 - 9	(3f, 7f)	(0f, 1f)	(03, 07)	(40, 01)	(10, 20)	(04, 08)	(41, 02)	(30, 60)	(0c, 18)	(43, 06)
10 - 19	(50, 21)	(14, 28)	(45, 0a)	(71, 62)	(3c, 78)	(4f, 1e)	(13, 27)	(44, 09)	(51, 22)	(34, 68)
20 - 29	(4d, 1a)	(66, 73)	(5c, 39)	(57, 2e)	(15, 2b)	(65, 4a)	(79, 72)	(3e, 7c)	(2f, 5f)	(0b, 17)
30 - 39	(42, 05)	(70, 61)	(1c, 38)	(47, 0e)	(11, 23)	(24, 48)	(49, 12)	(32, 64)	(6c, 59)	(5b, 36)
40 - 49	(56, 2d)	(35, 6b)	(6d, 5a)	(7b, 76)	(5e, 3d)	(37, 6f)	(0d, 1b)	(63, 46)	(58, 31)	(16, 2c)
50 - 59	(25, 4b)	(69, 52)	(74, 3a)	(6e, 5d)	(3b, 77)	(4e, 1d)	(33, 67)	(4c, 19)	(53, 26)	(54, 29)
60 - 69	(55, 2a)	(75, 6a)	(7d, 7a)	(7f, 7e)	(1f, 3f)	(07, 0f)	(01, 03)	(20, 40)	(08, 10)	(02, 04)
70 - 79	(60, 41)	(18, 30)	(06, 0c)	(21, 43)	(28, 50)	(0a, 14)	(62, 45)	(78, 71)	(1e, 3c)	(27, 4f)
80 - 89	(09, 13)	(22, 44)	(68, 51)	(1a, 34)	(66, 4d)	(39, 73)	(2e, 5c)	(2b, 57)	(4a, 15)	(72, 65)
90 - 99	(7c, 79)	(5f, 3e)	(17, 2f)	(05, 0b)	(61, 42)	(38, 70)	(0e, 1c)	(23, 47)	(48, 11)	(12, 24)
100 - 109	(64, 49)	(59, 32)	(36, 6c)	(2d, 5b)	(6b, 56)	(5a, 35)	(76, 6d)	(3d, 7b)	(6f, 5e)	(1b, 37)
110	(46, 0d)									

A.2 MILP Model for Computing Minimum Number of Active Sboxes

```
from gurobipy import *
import time

class wg :
  def __init__(self, rounds, ft, sut, mid, nw):
    self.rounds = rounds                     # Number of rounds
    self.ft = ft                         # Feedback taps including last sbox
    self.sut = sut                         # Sbox and updated taps (without last and middle wgp sbox)
    self.nw = nw                         # Number of words in LFSR
    self.mid = mid                         # Tap position of middle wgp sbox
    self.file_model = "wg_" + str(self.rounds)  + ".lp"
    self.file_binary = "wg_Binary" + str(self.rounds)  +".txt"
    fp = open(self.file_model,"w")
    fp.close()
    fp = open(self.file_binary,"w")
    fp.close()

  def create_variables(self, x, r, len):
    variable = []
    for i in range(len):
      variable.append(x + "_" + str(r) + "_" + str(i))
    return variable

  def create_dvariables(self, x, r):              # Dummy variable for XOR
    variable = []
    variable.append(x + "_" + str(r))
    return variable

  def xor_two_words(self, x_in0, x_in1, x_out, r, ind):   # XOR of two words
    D = self.create_dvariables('d'+str(ind), r)

    fp = open(self.file_binary, "a")
    fp.write(D[0] + str("\n")) ; fp.close() ;

    fp = open(self.file_model, "a")
    fp.write(x_in0 + " + " + x_in1 + " + " + x_out + " - 2 " + D[0] + " >= 0 \n")
    fp.write(D[0] + " - " + x_in0 + " >= 0\n")
    fp.write(D[0] + " - " + x_in1 + " >= 0\n")
    fp.write(D[0] + " - " + x_out + " >= 0\n")
    fp.close()

  def fb_xor(self, X, Y, r):                    # constraints for the feedback
    D = self.create_dvariables('d'+str(self.nw-1), r)
    fp = open(self.file_binary, "a")
    fp.write(D[0] + str("\n"))
    fp.close()

    fp = open(self.file_model, "a")
    temp = []

    for t in X:
      temp.append(t)

    temp.append(Y[0])
    temp = " + ".join(temp)
    temp = temp + " - 2 " + D[0] + " >= 0"
    fp.write(temp + "\n")

    for t in X:
      fp.write(D[0] + " - " + t + " >= 0\n")
    fp.write(D[0] + " - " + Y[0] + " >= 0\n")
    fp.close()

  def shift_state(self, X, Y):
    for i in range(self.nw - 1):
      X[i] = X[i+1]
    X[self.nw-1] = Y
    return X

  def state_update(self):
    X = self.create_variables('x', 0, self.nw)        # Initial state variables

    fp = open(self.file_binary, "a")
    for i in range(self.nw):
      fp.write(X[i] + "\n")
    fp.close()

    for i in range(self.rounds):

      Y = self.create_variables('x', i+1, self.nw)

      fp = open(self.file_binary,'a')
      fp.write(Y[-1] + "\n")
      for t in self.sut:
        fp.write(Y[t[1]] + "\n")
      fp.write(Y[self.mid+1] + "\n")
      fp.close()

      self.fb_xor([X[t] for t in self.ft], [Y[-1]], i)

      for t in self.sut:
        self.xor_two_words(X[t[0]], X[t[1]], Y[t[1]], i, t[1])

      self.xor_two_words(X[self.mid], X[self.mid + 1], Y[self.mid+1], i, self.mid
   +1)

      for t in self.sut:
        X[t[1]] = Y[t[1]]
      X[self.mid + 1] = Y[self.mid  + 1]
      X = self.shift_state(X, Y[-1])

  def init(self):

    fp = open(self.file_model, "a")
```

```
    fp.write("Minimize\n")

    temp = []
    X = self.create_variables('x', 0, self.nw)

    for t in self.sut:
      temp.append(X[t[0]])
    temp.append(X[self.mid])
    temp.append(X[self.nw-1])

    for i in range(self.rounds-1):
      Y = self.create_variables('x', i+1, self.nw)

      for t in self.sut:
        X[t[1]] = Y[t[1]]
      X[self.mid + 1] = Y[self.mid  + 1]
      X = self.shift_state(X, Y[-1])
      for t in self.sut:
        temp.append(X[t[0]])
      temp.append(X[self.mid])
      temp.append(X[self.nw-1])

    temp = " + ".join(temp)
    fp.write(temp+ "\n")
    fp.write("Subject To\n")

    temp = []
    for i in range(self.nw):
      temp.append('x_0_'+str(i))
    temp = " + ".join(temp)
    temp = temp + " >= 1 \n"
    fp.write(temp)

    fp.close()

  def make_model(self):
    self.init()
    self.state_update()
    fileobj = open(self.file_model, "a")
    fileobj1 = open(self.file_binary, "r")
    fileobj.write("Binary\n")
    for line in fileobj1:
      fileobj.write(line)
    fileobj.write("END\n")
    fileobj.close()
    fileobj1.close()
```

```

def solve_model(self):
  m = read(self.file_model)
  m.optimize()
  if(m.Status ==2):
    obj = m.getObjective()
    return (obj.getValue())
```

Appendix B

Appendices: Correlated Sequences

B.1 Data Structure

Consider a toy Simon-8/16 cipher as given in Example 9.1. Let $k = (1, 2, 3, 4)$ and $s_0^e = 15$, $s_1^e = 14$, then $s_0^d = 5$ and $s_1^d = 11$. In Table B.1, we provide the data structure DS^d that is used for 6 decryption rounds.

Table B.1: Data structure for toy Simon

i	$\mathsf{DS}^{\mathsf{d}}[i]$
0	[9, 15, [9, 14, 7, 2, 4, 3, 14, 11, 2, 4, 12, 8, 7, 1, 13, 9]]
1	[8, 10, [2, 5, 10, 15, 15, 8, 3, 6, 0, 6, 8, 12, 5, 3, 9, 13]]
2	[11, 5, [14, 10, 0, 6, 15, 11, 5, 3, 5, 0, 11, 12, 12, 9, 6, 1]]
3	[10, 2, [2, 4, 10, 14, 7, 1, 11, 15, 0, 7, 8, 13, 13, 10, 1, 4]]
4	[13, 6, [0, 6, 12, 8, 5, 3, 13, 9, 10, 13, 6, 3, 7, 0, 15, 10]]
5	[12, 3, [2, 4, 8, 12, 7, 1, 9, 13, 1, 6, 11, 14, 12, 11, 2, 7]]
6	[15, 8, [13, 8, 5, 2, 4, 1, 8, 15, 15, 11, 7, 1, 14, 10, 2, 4]]
7	[14, 15, [14, 9, 0, 5, 3, 4, 9, 12, 5, 3, 11, 15, 0, 6, 10, 14]]
8	[1, 12, [10, 15, 6, 1, 3, 6, 11, 12, 0, 4, 12, 10, 1, 5, 9, 15]]
9	[0, 8, [2, 7, 10, 13, 11, 14, 7, 0, 0, 4, 8, 14, 1, 5, 13, 11]]
10	[3, 6, [14, 8, 2, 6, 11, 13, 3, 7, 4, 3, 8, 13, 9, 14, 1, 4]]
11	[2, 0, [2, 6, 10, 12, 3, 7, 15, 9, 0, 5, 8, 15, 9, 12, 5, 2]]
12	[5, 13, [11, 14, 5, 2, 2, 7, 8, 15, 0, 4, 14, 8, 1, 5, 11, 13]]
13	[4, 9, [3, 6, 9, 14, 10, 15, 4, 3, 0, 4, 10, 12, 1, 5, 15, 9]]
14	[7, 3, [9, 15, 3, 7, 12, 10, 2, 6, 10, 13, 0, 5, 7, 0, 9, 12]]
15	[6, 5, [3, 7, 13, 11, 2, 6, 8, 14, 8, 13, 6, 1, 1, 4, 11, 12]]
16	[12, 2, [4, 2, 12, 8, 1, 7, 13, 9, 6, 1, 14, 11, 11, 12, 7, 2]]
17	[13, 7, [6, 0, 8, 12, 3, 5, 9, 13, 13, 10, 3, 6, 0, 7, 10, 15]]
18	[14, 14, [9, 14, 5, 0, 4, 3, 12, 9, 3, 5, 15, 11, 6, 0, 14, 10]]
19	[15, 9, [8, 13, 2, 5, 1, 4, 15, 8, 11, 15, 1, 7, 10, 14, 4, 2]]
20	[8, 11, [5, 2, 15, 10, 8, 15, 6, 3, 6, 0, 12, 8, 3, 5, 13, 9]]
21	[9, 14, [14, 9, 2, 7, 3, 4, 11, 14, 4, 2, 8, 12, 1, 7, 9, 13]]
22	[10, 3, [4, 2, 14, 10, 1, 7, 15, 11, 7, 0, 13, 8, 10, 13, 4, 1]]
23	[11, 4, [10, 14, 6, 0, 11, 15, 3, 5, 0, 5, 12, 11, 9, 12, 1, 6]]
24	[4, 8, [6, 3, 14, 9, 15, 10, 3, 4, 4, 0, 12, 10, 5, 1, 9, 15]]
25	[5, 12, [14, 11, 2, 5, 7, 2, 15, 8, 4, 0, 8, 14, 5, 1, 13, 11]]
26	[6, 4, [7, 3, 11, 13, 6, 2, 14, 8, 13, 8, 1, 6, 4, 1, 12, 11]]
27	[7, 2, [15, 9, 7, 3, 10, 12, 6, 2, 13, 10, 5, 0, 0, 7, 12, 9]]
28	[0, 9, [7, 2, 13, 10, 14, 11, 0, 7, 4, 0, 14, 8, 5, 1, 11, 13]]
29	[1, 13, [15, 10, 1, 6, 6, 3, 12, 11, 4, 0, 10, 12, 5, 1, 15, 9]]
30	[2, 1, [6, 2, 12, 10, 7, 3, 9, 15, 5, 0, 15, 8, 12, 9, 2, 5]]
31	[3, 7, [8, 14, 6, 2, 13, 11, 7, 3, 3, 4, 13, 8, 14, 9, 4, 1]]
32	[1, 14, [6, 1, 10, 15, 11, 12, 3, 6, 12, 10, 0, 4, 9, 15, 1, 5]]
33	[0, 10, [10, 13, 2, 7, 7, 0, 11, 14, 8, 14, 0, 4, 13, 11, 1, 5]]
34	[3, 4, [2, 6, 14, 8, 3, 7, 11, 13, 8, 13, 4, 3, 1, 4, 9, 14]]
35	[2, 2, [10, 12, 2, 6, 15, 9, 3, 7, 8, 15, 0, 5, 5, 2, 9, 12]]
36	[5, 15, [5, 2, 11, 14, 8, 15, 2, 7, 14, 8, 0, 4, 11, 13, 1, 5]]
37	[4, 11, [9, 14, 3, 6, 4, 3, 10, 15, 10, 12, 0, 4, 15, 9, 1, 5]]
38	[7, 1, [3, 7, 9, 15, 2, 6, 12, 10, 0, 5, 10, 13, 9, 12, 7, 0]]
39	[6, 7, [13, 11, 3, 7, 8, 14, 2, 6, 6, 1, 8, 13, 11, 12, 1, 4]]
40	[9, 13, [7, 2, 9, 14, 14, 11, 4, 3, 12, 8, 2, 4, 13, 9, 7, 1]]
41	[8, 8, [10, 15, 2, 5, 3, 6, 15, 8, 8, 12, 0, 6, 9, 13, 5, 3]]
42	[11, 7, [0, 6, 14, 10, 5, 3, 15, 11, 11, 12, 5, 0, 6, 1, 12, 9]]
43	[10, 0, [10, 14, 2, 4, 11, 15, 7, 1, 8, 13, 0, 7, 1, 4, 13, 10]]
44	[13, 4, [12, 8, 0, 6, 13, 9, 5, 3, 6, 3, 10, 13, 15, 10, 7, 0]]
45	[12, 1, [8, 12, 2, 4, 9, 13, 7, 1, 11, 14, 1, 6, 2, 7, 12, 11]]
46	[15, 10, [5, 2, 13, 8, 8, 15, 4, 1, 7, 1, 15, 11, 2, 4, 14, 10]]
47	[14, 13, [0, 5, 14, 9, 9, 12, 3, 4, 11, 15, 5, 3, 10, 14, 0, 6]]
48	[6, 6, [11, 13, 7, 3, 14, 8, 6, 2, 1, 6, 13, 8, 12, 11, 4, 1]]
49	[7, 0, [7, 3, 15, 9, 6, 2, 10, 12, 5, 0, 13, 10, 12, 9, 0, 7]]
50	[4, 10, [14, 9, 6, 3, 3, 4, 15, 10, 12, 10, 4, 0, 9, 15, 5, 1]]
51	[5, 14, [2, 5, 14, 11, 15, 8, 7, 2, 8, 14, 4, 0, 13, 11, 5, 1]]
52	[2, 3, [12, 10, 6, 2, 9, 15, 7, 3, 15, 8, 5, 0, 2, 5, 12, 9]]
53	[3, 5, [6, 2, 8, 14, 7, 3, 13, 11, 13, 8, 3, 4, 4, 1, 14, 9]]
54	[0, 11, [13, 10, 7, 2, 0, 7, 14, 11, 14, 8, 4, 0, 11, 13, 5, 1]]
55	[1, 15, [1, 6, 15, 10, 12, 11, 6, 3, 10, 12, 4, 0, 15, 9, 5, 1]]
56	[14, 12, [5, 0, 9, 14, 12, 9, 4, 3, 15, 11, 3, 5, 14, 10, 6, 0]]
57	[15, 11, [2, 5, 8, 13, 15, 8, 1, 4, 1, 7, 11, 15, 4, 2, 10, 14]]
58	[12, 0, [12, 8, 4, 2, 13, 9, 1, 7, 14, 11, 6, 1, 7, 2, 11, 12]]
59	[13, 5, [8, 12, 6, 0, 9, 13, 3, 5, 3, 6, 13, 10, 10, 15, 0, 7]]
60	[10, 1, [14, 10, 4, 2, 15, 11, 1, 7, 13, 8, 7, 0, 4, 1, 10, 13]]
61	[11, 6, [6, 0, 10, 14, 3, 5, 11, 15, 12, 11, 0, 5, 1, 6, 9, 12]]
62	[8, 9, [15, 10, 5, 2, 6, 3, 8, 15, 12, 8, 6, 0, 13, 9, 3, 5]]
63	[9, 12, [2, 7, 14, 9, 11, 14, 3, 4, 8, 12, 4, 2, 9, 13, 1, 7]]
64	[0, 12, [11, 14, 7, 0, 2, 7, 10, 13, 1, 5, 13, 11, 0, 4, 8, 14]]
65	[1, 8, [3, 6, 11, 12, 10, 15, 6, 1, 1, 5, 9, 15, 0, 4, 12, 10]]
66	[2, 4, [3, 7, 15, 9, 2, 6, 10, 12, 9, 12, 5, 2, 0, 5, 8, 15]]
67	[3, 2, [11, 13, 3, 7, 14, 8, 2, 6, 9, 14, 1, 4, 4, 3, 8, 13]]
68	[4, 13, [10, 15, 4, 3, 3, 6, 9, 14, 1, 5, 15, 9, 0, 4, 10, 12]]
69	[5, 9, [2, 7, 8, 15, 11, 14, 5, 2, 1, 5, 11, 13, 0, 4, 14, 8]]
70	[6, 1, [2, 6, 8, 14, 3, 7, 13, 11, 1, 4, 11, 12, 8, 13, 6, 1]]
71	[7, 7, [12, 10, 2, 6, 9, 15, 3, 7, 7, 0, 9, 12, 10, 13, 0, 5]]
72	[8, 14, [15, 8, 3, 6, 2, 5, 10, 15, 5, 3, 9, 13, 0, 6, 8, 12]]
73	[9, 11, [4, 3, 14, 11, 9, 14, 7, 2, 7, 1, 13, 9, 2, 4, 12, 8]]
74	[10, 6, [7, 1, 11, 15, 2, 4, 10, 14, 13, 10, 1, 4, 0, 7, 8, 13]]
75	[11, 1, [15, 11, 5, 3, 14, 10, 0, 6, 12, 9, 6, 1, 5, 0, 11, 12]]
76	[12, 7, [7, 1, 9, 13, 2, 4, 8, 12, 12, 11, 2, 7, 1, 6, 11, 14]]
77	[13, 2, [5, 3, 13, 9, 0, 6, 12, 8, 7, 0, 15, 10, 10, 13, 6, 3]]
78	[14, 11, [3, 4, 9, 12, 14, 9, 0, 5, 0, 6, 10, 14, 5, 3, 11, 15]]
79	[15, 12, [4, 1, 8, 15, 13, 8, 5, 2, 14, 10, 2, 4, 15, 11, 7, 1]]
80	[5, 8, [7, 2, 15, 8, 14, 11, 2, 5, 5, 1, 13, 11, 4, 0, 8, 14]]
81	[4, 12, [15, 10, 3, 4, 6, 3, 14, 9, 5, 1, 9, 15, 4, 0, 12, 10]]
82	[7, 6, [10, 12, 6, 2, 15, 9, 7, 3, 0, 7, 12, 9, 13, 10, 5, 0]]
83	[6, 0, [6, 2, 14, 8, 7, 3, 11, 13, 4, 1, 12, 11, 13, 8, 1, 6]]
84	[1, 9, [6, 3, 12, 11, 15, 10, 1, 6, 5, 1, 15, 9, 4, 0, 10, 12]]
85	[0, 13, [14, 11, 0, 7, 7, 2, 13, 10, 5, 1, 11, 13, 4, 0, 14, 8]]
86	[3, 3, [13, 11, 7, 3, 8, 14, 6, 2, 14, 9, 4, 1, 3, 4, 13, 8]]
87	[2, 5, [7, 3, 9, 15, 6, 2, 12, 10, 12, 9, 2, 5, 5, 0, 15, 8]]
88	[13, 3, [3, 5, 9, 13, 6, 0, 8, 12, 0, 7, 10, 15, 13, 10, 3, 6]]
89	[12, 6, [1, 7, 13, 9, 4, 2, 12, 8, 11, 12, 7, 2, 6, 1, 14, 11]]
90	[15, 13, [1, 4, 15, 8, 8, 13, 2, 5, 10, 14, 4, 2, 11, 15, 1, 7]]
91	[14, 10, [4, 3, 12, 9, 9, 14, 5, 0, 6, 0, 14, 10, 3, 5, 15, 11]]
92	[9, 10, [3, 4, 11, 14, 14, 9, 2, 7, 1, 7, 9, 13, 4, 2, 8, 12]]
93	[8, 15, [8, 15, 6, 3, 5, 2, 15, 10, 3, 5, 13, 9, 6, 0, 12, 8]]
94	[11, 0, [11, 15, 3, 5, 10, 14, 6, 0, 9, 12, 1, 6, 0, 5, 12, 11]]
95	[10, 7, [1, 7, 15, 11, 4, 2, 14, 10, 10, 13, 4, 1, 7, 0, 13, 8]]
96	[12, 5, [9, 13, 7, 1, 8, 12, 2, 4, 2, 7, 12, 11, 11, 14, 1, 6]]
97	[13, 0, [13, 9, 5, 3, 12, 8, 0, 6, 15, 10, 7, 0, 6, 3, 10, 13]]
98	[14, 9, [9, 12, 3, 4, 0, 5, 14, 9, 10, 14, 0, 6, 11, 15, 5, 3]]
99	[15, 14, [8, 15, 4, 1, 5, 2, 13, 8, 2, 4, 14, 10, 7, 1, 15, 11]]
100	[8, 12, [3, 6, 15, 8, 10, 15, 2, 5, 9, 13, 5, 3, 8, 12, 0, 6]]
101	[9, 9, [14, 11, 4, 3, 7, 2, 9, 14, 13, 9, 7, 1, 12, 8, 2, 4]]
102	[10, 4, [11, 15, 7, 1, 10, 14, 2, 4, 1, 4, 13, 10, 8, 13, 0, 7]]
103	[11, 3, [5, 3, 15, 11, 0, 6, 14, 10, 6, 1, 12, 9, 11, 12, 5, 0]]
104	[4, 15, [4, 3, 10, 15, 9, 14, 3, 6, 15, 9, 1, 5, 10, 12, 0, 4]]
105	[5, 11, [8, 15, 2, 7, 5, 2, 11, 14, 11, 13, 1, 5, 14, 8, 0, 4]]
106	[6, 3, [8, 14, 2, 6, 13, 11, 3, 7, 11, 12, 1, 4, 6, 1, 8, 13]]
107	[7, 5, [2, 6, 12, 10, 3, 7, 9, 15, 9, 12, 7, 0, 0, 5, 10, 13]]
108	[0, 14, [7, 0, 11, 14, 10, 13, 2, 7, 13, 11, 1, 5, 8, 14, 0, 4]]
109	[1, 10, [11, 12, 3, 6, 6, 1, 10, 15, 9, 15, 1, 5, 12, 10, 0, 4]]
110	[2, 6, [15, 9, 3, 7, 10, 12, 2, 6, 5, 2, 9, 12, 8, 15, 0, 5]]
111	[3, 0, [3, 7, 11, 13, 2, 6, 14, 8, 1, 4, 9, 14, 8, 13, 4, 3]]
112	[11, 2, [3, 5, 11, 15, 6, 0, 10, 14, 1, 6, 9, 12, 12, 11, 0, 5]]
113	[10, 5, [15, 11, 1, 7, 14, 10, 4, 2, 4, 1, 10, 13, 13, 8, 7, 0]]
114	[9, 8, [11, 14, 3, 4, 2, 7, 14, 9, 9, 13, 1, 7, 8, 12, 4, 2]]
115	[8, 13, [6, 3, 8, 15, 15, 10, 5, 2, 13, 9, 3, 5, 12, 8, 6, 0]]
116	[15, 15, [15, 8, 1, 4, 2, 5, 8, 13, 4, 2, 10, 14, 1, 7, 11, 15]]
117	[14, 8, [12, 9, 4, 3, 5, 0, 9, 14, 14, 10, 6, 0, 15, 11, 3, 5]]
118	[13, 1, [9, 13, 3, 5, 8, 12, 6, 0, 10, 15, 0, 7, 3, 6, 13, 10]]
119	[12, 4, [13, 9, 1, 7, 12, 8, 4, 2, 7, 2, 11, 12, 14, 11, 6, 1]]
120	[3, 1, [7, 3, 13, 11, 6, 2, 8, 14, 4, 1, 14, 9, 13, 8, 3, 4]]
121	[2, 7, [9, 15, 7, 3, 12, 10, 6, 2, 2, 5, 12, 9, 15, 8, 5, 0]]
122	[1, 11, [12, 11, 6, 3, 1, 6, 15, 10, 15, 9, 5, 1, 10, 12, 4, 0]]
123	[0, 15, [0, 7, 14, 11, 13, 10, 7, 2, 11, 13, 5, 1, 14, 8, 4, 0]]
124	[7, 4, [6, 2, 10, 12, 7, 3, 15, 9, 12, 9, 0, 7, 5, 0, 13, 10]]
125	[6, 2, [14, 8, 6, 2, 11, 13, 7, 3, 12, 11, 4, 1, 1, 6, 13, 8]]
126	[5, 10, [15, 8, 7, 2, 2, 5, 14, 11, 13, 11, 5, 1, 8, 14, 4, 0]]
127	[4, 14, [3, 4, 15, 10, 14, 9, 6, 3, 9, 15, 5, 1, 12, 10, 4, 0]]
128	[11, 13, [5, 0, 11, 12, 12, 9, 6, 1, 14, 10, 0, 6, 15, 11, 5, 3]]

i	$\mathsf{DS}^{\mathsf{d}}[i]$
129	[10, 10, [0, 7, 8, 13, 13, 10, 1, 4, 2, 4, 10, 14, 7, 1, 11, 15]]
130	[9, 7, [2, 4, 12, 8, 7, 1, 13, 9, 9, 14, 7, 2, 4, 3, 14, 11]]
131	[8, 2, [0, 6, 8, 12, 5, 3, 9, 13, 2, 5, 10, 15, 15, 8, 3, 6]]
132	[15, 0, [15, 11, 7, 1, 14, 10, 2, 4, 13, 8, 5, 2, 4, 1, 8, 15]]
133	[14, 7, [5, 3, 11, 15, 0, 6, 10, 14, 14, 9, 0, 5, 3, 4, 9, 12]]
134	[13, 14, [10, 13, 6, 3, 7, 0, 15, 10, 0, 6, 12, 8, 5, 3, 13, 9]]
135	[12, 11, [1, 6, 11, 14, 12, 11, 2, 7, 2, 4, 8, 12, 7, 1, 9, 13]]
136	[3, 14, [4, 3, 8, 13, 9, 14, 1, 4, 14, 8, 2, 6, 11, 13, 3, 7]]
137	[2, 8, [0, 5, 8, 15, 9, 12, 5, 2, 2, 6, 10, 12, 3, 7, 15, 9]]
138	[1, 4, [0, 4, 12, 10, 1, 5, 9, 15, 10, 15, 6, 1, 3, 6, 11, 12]]
139	[0, 0, [0, 4, 8, 14, 1, 5, 13, 11, 2, 7, 10, 13, 11, 14, 7, 0]]
140	[7, 11, [10, 13, 0, 5, 7, 0, 9, 12, 9, 15, 3, 7, 12, 10, 2, 6]]
141	[6, 13, [8, 13, 6, 1, 1, 4, 11, 12, 3, 7, 13, 11, 2, 6, 8, 14]]
142	[5, 5, [0, 4, 14, 8, 1, 5, 11, 13, 11, 14, 5, 2, 2, 7, 8, 15]]
143	[4, 1, [0, 4, 10, 12, 1, 5, 15, 9, 3, 6, 9, 14, 10, 15, 4, 3]]
144	[15, 1, [11, 15, 1, 7, 10, 14, 4, 2, 8, 13, 2, 5, 1, 4, 15, 8]]
145	[14, 6, [3, 5, 15, 11, 6, 0, 14, 10, 9, 14, 5, 0, 4, 3, 12, 9]]
146	[13, 15, [13, 10, 3, 6, 0, 7, 10, 15, 6, 0, 8, 12, 3, 5, 9, 13]]
147	[12, 10, [6, 1, 14, 11, 11, 12, 7, 2, 4, 2, 12, 8, 1, 7, 13, 9]]
148	[11, 12, [0, 5, 12, 11, 9, 12, 1, 6, 10, 14, 6, 0, 11, 15, 3, 5]]
149	[10, 11, [7, 0, 13, 8, 10, 13, 4, 1, 4, 2, 14, 10, 1, 7, 15, 11]]
150	[9, 6, [4, 2, 8, 12, 1, 7, 9, 13, 14, 9, 2, 7, 3, 4, 11, 14]]
151	[8, 3, [6, 0, 12, 8, 3, 5, 13, 9, 5, 2, 15, 10, 8, 15, 6, 3]]
152	[7, 10, [13, 10, 5, 0, 0, 7, 12, 9, 15, 9, 7, 3, 10, 12, 6, 2]]
153	[6, 12, [13, 8, 1, 6, 4, 1, 12, 11, 7, 3, 11, 13, 6, 2, 14, 8]]
154	[5, 4, [4, 0, 8, 14, 5, 1, 13, 11, 14, 11, 2, 5, 7, 2, 15, 8]]
155	[4, 0, [4, 0, 12, 10, 5, 1, 9, 15, 6, 3, 14, 9, 15, 10, 3, 4]]
156	[3, 15, [3, 4, 13, 8, 14, 9, 4, 1, 8, 14, 6, 2, 13, 11, 7, 3]]
157	[2, 9, [5, 0, 15, 8, 12, 9, 2, 5, 6, 2, 12, 10, 7, 3, 9, 15]]
158	[1, 5, [4, 0, 10, 12, 5, 1, 15, 9, 15, 10, 1, 6, 6, 3, 12, 11]]
159	[0, 1, [4, 0, 14, 8, 5, 1, 11, 13, 7, 2, 13, 10, 14, 11, 0, 7]]
160	[3, 12, [8, 13, 4, 3, 1, 4, 9, 14, 2, 6, 14, 8, 3, 7, 11, 13]]
161	[2, 10, [8, 15, 0, 5, 5, 2, 9, 12, 10, 12, 2, 6, 15, 9, 3, 7]]
162	[1, 6, [12, 10, 0, 4, 9, 15, 1, 5, 6, 1, 10, 15, 11, 12, 3, 6]]
163	[0, 2, [8, 14, 0, 4, 13, 11, 1, 5, 10, 13, 2, 7, 7, 0, 11, 14]]
164	[7, 9, [0, 5, 10, 13, 9, 12, 7, 0, 3, 7, 9, 15, 2, 6, 12, 10]]
165	[6, 15, [6, 1, 8, 13, 11, 12, 1, 4, 13, 11, 3, 7, 8, 14, 2, 6]]
166	[5, 7, [14, 8, 0, 4, 11, 13, 1, 5, 5, 2, 11, 14, 8, 15, 2, 7]]
167	[4, 3, [10, 12, 0, 4, 15, 9, 1, 5, 9, 14, 3, 6, 4, 3, 10, 15]]
168	[11, 15, [11, 12, 5, 0, 6, 1, 12, 9, 0, 6, 14, 10, 5, 3, 15, 11]]
169	[10, 8, [8, 13, 0, 7, 1, 4, 13, 10, 10, 14, 2, 4, 11, 15, 7, 1]]
170	[9, 5, [12, 8, 2, 4, 13, 9, 7, 1, 7, 2, 9, 14, 14, 11, 4, 3]]
171	[8, 0, [8, 12, 0, 6, 9, 13, 5, 3, 10, 15, 2, 5, 3, 6, 15, 8]]
172	[15, 2, [7, 1, 15, 11, 2, 4, 14, 10, 5, 2, 13, 8, 8, 15, 4, 1]]
173	[14, 5, [11, 15, 5, 3, 10, 14, 0, 6, 0, 5, 14, 9, 9, 12, 3, 4]]
174	[13, 12, [6, 3, 10, 13, 15, 10, 7, 0, 12, 8, 0, 6, 13, 9, 5, 3]]
175	[12, 9, [11, 14, 1, 6, 2, 7, 12, 11, 8, 12, 2, 4, 9, 13, 7, 1]]
176	[5, 6, [8, 14, 4, 0, 13, 11, 5, 1, 2, 5, 14, 11, 15, 8, 7, 2]]
177	[4, 2, [12, 10, 4, 0, 9, 15, 5, 1, 14, 9, 6, 3, 3, 4, 15, 10]]
178	[7, 8, [5, 0, 13, 10, 12, 9, 0, 7, 7, 3, 15, 9, 6, 2, 10, 12]]
179	[6, 14, [1, 6, 13, 8, 12, 11, 4, 1, 11, 13, 7, 3, 14, 8, 6, 2]]
180	[1, 7, [10, 12, 4, 0, 15, 9, 5, 1, 1, 6, 15, 10, 12, 11, 6, 3]]
181	[0, 3, [14, 8, 4, 0, 11, 13, 5, 1, 13, 10, 7, 2, 0, 7, 14, 11]]
182	[3, 13, [13, 8, 3, 4, 4, 1, 14, 9, 6, 2, 8, 14, 7, 3, 13, 11]]
183	[2, 11, [15, 8, 5, 0, 2, 5, 12, 9, 12, 10, 6, 2, 9, 15, 7, 3]]
184	[13, 13, [3, 6, 13, 10, 10, 15, 0, 7, 8, 12, 6, 0, 9, 13, 3, 5]]
185	[12, 8, [14, 11, 6, 1, 7, 2, 11, 12, 12, 8, 4, 2, 13, 9, 1, 7]]
186	[15, 3, [1, 7, 11, 15, 4, 2, 10, 14, 2, 5, 8, 13, 15, 8, 1, 4]]
187	[14, 4, [15, 11, 3, 5, 14, 10, 6, 0, 5, 0, 9, 14, 12, 9, 4, 3]]
188	[9, 4, [8, 12, 4, 2, 9, 13, 1, 7, 2, 7, 14, 9, 11, 14, 3, 4]]
189	[8, 1, [12, 8, 6, 0, 13, 9, 3, 5, 15, 10, 5, 2, 6, 3, 8, 15]]
190	[11, 14, [12, 11, 0, 5, 1, 6, 9, 12, 6, 0, 10, 14, 3, 5, 11, 15]]
191	[10, 9, [13, 8, 7, 0, 4, 1, 10, 13, 14, 10, 4, 2, 15, 11, 1, 7]]
192	[10, 14, [13, 10, 1, 4, 0, 7, 8, 13, 7, 1, 11, 15, 2, 4, 10, 14]]
193	[11, 9, [12, 9, 6, 1, 5, 0, 11, 12, 15, 11, 5, 3, 14, 10, 0, 6]]
194	[8, 6, [5, 3, 9, 13, 0, 6, 8, 12, 15, 8, 3, 6, 2, 5, 10, 15]]
195	[9, 3, [7, 1, 13, 9, 2, 4, 12, 8, 4, 3, 14, 11, 9, 14, 7, 2]]
196	[14, 3, [0, 6, 10, 14, 5, 3, 11, 15, 3, 4, 9, 12, 14, 9, 0, 5]]
197	[15, 4, [14, 10, 2, 4, 15, 11, 7, 1, 4, 1, 8, 15, 13, 8, 5, 2]]
198	[12, 15, [12, 11, 2, 7, 1, 6, 11, 14, 7, 1, 9, 13, 2, 4, 8, 12]]
199	[13, 10, [7, 0, 15, 10, 10, 13, 6, 3, 5, 3, 13, 9, 0, 6, 12, 8]]
200	[2, 12, [9, 12, 5, 2, 0, 5, 8, 15, 3, 7, 15, 9, 2, 6, 10, 12]]
201	[3, 10, [9, 14, 1, 4, 4, 3, 8, 13, 11, 13, 3, 7, 14, 8, 2, 6]]
202	[0, 4, [1, 5, 13, 11, 0, 4, 8, 14, 11, 14, 7, 0, 2, 7, 10, 13]]
203	[1, 0, [1, 5, 9, 15, 0, 4, 12, 10, 3, 6, 11, 12, 10, 15, 6, 1]]
204	[6, 9, [1, 4, 11, 12, 8, 13, 6, 1, 2, 6, 8, 14, 3, 7, 13, 11]]
205	[7, 15, [7, 0, 9, 12, 10, 13, 0, 5, 12, 10, 2, 6, 9, 15, 3, 7]]
206	[4, 5, [1, 5, 15, 9, 0, 4, 10, 12, 10, 15, 4, 3, 3, 6, 9, 14]]
207	[5, 1, [1, 5, 11, 13, 0, 4, 14, 8, 2, 7, 8, 15, 11, 14, 5, 2]]
208	[14, 2, [6, 0, 14, 10, 3, 5, 15, 11, 4, 3, 12, 9, 9, 14, 5, 0]]
209	[15, 5, [10, 14, 4, 2, 11, 15, 1, 7, 1, 4, 15, 8, 8, 13, 2, 5]]
210	[12, 14, [11, 12, 7, 2, 6, 1, 14, 11, 1, 7, 13, 9, 4, 2, 12, 8]]
211	[13, 11, [0, 7, 10, 15, 13, 10, 3, 6, 3, 5, 9, 13, 6, 0, 8, 12]]
212	[10, 15, [10, 13, 4, 1, 7, 0, 13, 8, 1, 7, 15, 11, 4, 2, 14, 10]]
213	[11, 8, [9, 12, 1, 6, 0, 5, 12, 11, 11, 15, 3, 5, 10, 14, 6, 0]]
214	[8, 7, [3, 5, 13, 9, 6, 0, 12, 8, 8, 15, 6, 3, 5, 2, 15, 10]]
215	[9, 2, [1, 7, 9, 13, 4, 2, 8, 12, 3, 4, 11, 14, 14, 9, 2, 7]]
216	[6, 8, [4, 1, 12, 11, 13, 8, 1, 6, 6, 2, 14, 8, 7, 3, 11, 13]]
217	[7, 14, [0, 7, 12, 9, 13, 10, 5, 0, 10, 12, 6, 2, 15, 9, 7, 3]]
218	[4, 4, [5, 1, 9, 15, 4, 0, 12, 10, 15, 10, 3, 4, 6, 3, 14, 9]]
219	[5, 0, [5, 1, 13, 11, 4, 0, 8, 14, 7, 2, 15, 8, 14, 11, 2, 5]]
220	[2, 13, [12, 9, 2, 5, 5, 0, 15, 8, 7, 3, 9, 15, 6, 2, 12, 10]]
221	[3, 11, [14, 9, 4, 1, 3, 4, 13, 8, 13, 11, 7, 3, 8, 14, 6, 2]]
222	[0, 5, [5, 1, 11, 13, 4, 0, 14, 8, 14, 11, 0, 7, 7, 2, 13, 10]]
223	[1, 1, [5, 1, 15, 9, 4, 0, 10, 12, 6, 3, 12, 11, 15, 10, 1, 6]]
224	[6, 11, [11, 12, 1, 4, 6, 1, 8, 13, 8, 14, 2, 6, 13, 11, 3, 7]]
225	[7, 13, [9, 12, 7, 0, 0, 5, 10, 13, 2, 6, 12, 10, 3, 7, 9, 15]]
226	[4, 7, [15, 9, 1, 5, 10, 12, 0, 4, 4, 3, 10, 15, 9, 14, 3, 6]]
227	[5, 3, [11, 13, 1, 5, 14, 8, 0, 4, 8, 15, 2, 7, 5, 2, 11, 14]]
228	[2, 14, [5, 2, 9, 12, 8, 15, 0, 5, 15, 9, 3, 7, 10, 12, 2, 6]]
229	[3, 8, [1, 4, 9, 14, 8, 13, 4, 3, 3, 7, 11, 13, 2, 6, 14, 8]]
230	[0, 6, [13, 11, 1, 5, 8, 14, 0, 4, 7, 0, 11, 14, 10, 13, 2, 7]]
231	[1, 2, [9, 15, 1, 5, 12, 10, 0, 4, 11, 12, 3, 6, 6, 1, 10, 15]]
232	[14, 1, [10, 14, 0, 6, 11, 15, 5, 3, 9, 12, 3, 4, 0, 5, 14, 9]]
233	[15, 6, [2, 4, 14, 10, 7, 1, 15, 11, 8, 15, 4, 1, 5, 2, 13, 8]]
234	[12, 13, [2, 7, 12, 11, 11, 14, 1, 6, 9, 13, 7, 1, 8, 12, 2, 4]]
235	[13, 8, [15, 10, 7, 0, 6, 3, 10, 13, 13, 9, 5, 3, 12, 8, 0, 6]]
236	[10, 12, [1, 4, 13, 10, 8, 13, 0, 7, 11, 15, 7, 1, 10, 14, 2, 4]]
237	[11, 11, [6, 1, 12, 9, 11, 12, 5, 0, 5, 3, 15, 11, 0, 6, 14, 10]]
238	[8, 4, [9, 13, 5, 3, 8, 12, 0, 6, 3, 6, 15, 8, 10, 15, 2, 5]]
239	[9, 1, [13, 9, 7, 1, 12, 8, 2, 4, 14, 11, 4, 3, 7, 2, 9, 14]]
240	[0, 7, [11, 13, 5, 1, 14, 8, 4, 0, 0, 7, 14, 11, 13, 10, 7, 2]]
241	[1, 3, [15, 9, 5, 1, 10, 12, 4, 0, 12, 11, 6, 3, 1, 6, 15, 10]]
242	[2, 15, [2, 5, 12, 9, 15, 8, 5, 0, 9, 15, 7, 3, 12, 10, 6, 2]]
243	[3, 9, [4, 1, 14, 9, 13, 8, 3, 4, 7, 3, 13, 11, 6, 2, 8, 14]]
244	[4, 6, [9, 15, 5, 1, 12, 10, 4, 0, 3, 4, 15, 10, 14, 9, 6, 3]]
245	[5, 2, [13, 11, 5, 1, 8, 14, 4, 0, 15, 8, 7, 2, 2, 5, 14, 11]]
246	[6, 10, [12, 11, 4, 1, 1, 6, 13, 8, 14, 8, 6, 2, 11, 13, 7, 3]]
247	[7, 12, [12, 9, 0, 7, 5, 0, 13, 10, 6, 2, 10, 12, 7, 3, 15, 9]]
248	[8, 5, [13, 9, 3, 5, 12, 8, 6, 0, 6, 3, 8, 15, 15, 10, 5, 2]]
249	[9, 0, [9, 13, 1, 7, 8, 12, 4, 2, 11, 14, 3, 4, 2, 7, 14, 9]]
250	[10, 13, [4, 1, 10, 13, 13, 8, 7, 0, 15, 11, 1, 7, 14, 10, 4, 2]]
251	[11, 10, [1, 6, 9, 12, 12, 11, 0, 5, 3, 5, 11, 15, 6, 0, 10, 14]]
252	[12, 12, [7, 2, 11, 12, 14, 11, 6, 1, 13, 9, 1, 7, 12, 8, 4, 2]]
253	[13, 9, [10, 15, 0, 7, 3, 6, 13, 10, 9, 13, 3, 5, 8, 12, 6, 0]]
254	[14, 0, [14, 10, 6, 0, 15, 11, 3, 5, 12, 9, 4, 3, 5, 0, 9, 14]]
255	[15, 7, [4, 2, 10, 14, 1, 7, 11, 15, 15, 8, 1, 4, 2, 5, 8, 13]]

B.2 Computing the Eighth Element of the Sequence

Consider the parameters as given in Appendix B.1. The 25-round sequence is given in Table B.2.

Table B.2: 25-round sequence

i	0	1	2	3	4	5	6	7	8	9	10	11	12	-
k_i^e	1	2	3	4	3	0	8	10	1	3	15	15	7	-
s_i^e	15	14	9	11	7	4	5	1	9	12	3	1	8	-
i	13	14	15	16	17	18	19	20	21	22	23	24	25	26
k_i^e	6	8	6	4	11	13	11	2	3	3	1	7	-	-
s_i^e	12	4	11	1	9	2	10	5	4	6	10	15	11	5

We compute s_7^d using the data structure given in Table B.1 in the following 3 steps.

1. Find the row corresponding to (k_0^d, k_1^d) in $\mathsf{DS^d}$. The value of row is given by:

$$\begin{aligned} \mathsf{row} &= \mathsf{L}^4(k_0^d)||k_1^d \\ &= 7||1 = 113 \text{ (as } k_0^d = k_{24}^e = 7 \text{ and } k_1^d = k_{23}^e = 1) \end{aligned}$$

2. Compute the partition $p = \mathsf{DS^d}[113][2][k_2^d]+k_3^d = \mathsf{DS^d}[113][2][3]+3 = \mathsf{DS^d}[113][2][3]+3 = 7+3 = 4$. Note that $\mathsf{DS^d}[113][0][0] = s_3^d = s_{23}^e = 10$ and $X^d = 5$.

3. Compute s_7^d. We have

$$\begin{aligned} s_7^d &= s_3^d + k_5^d + p + \mathsf{DS^d}[113][2][s_6^d + X^d] \\ &= 10 + 11 + 4 + \mathsf{DS^d}[113][2][5+5] \\ &= 5 + \mathsf{DS^d}[113][2][0] = 5 + 15 = 10 = s_{19}^e. \end{aligned}$$