Advanced Techniques in Symmetric Key Cryptanalysis

A thesis submitted to Indian Statistical Institute
in partial fulfillment of the thesis requirements for the degree of
Doctor of Philosophy in Computer Science

Author: **Debasmita Chakraborty**

under the guidance of

Prof. Mridul Nandi

Applied Statistics Unit
Indian Statistical Institute, Kolkata
203 Barrackpore Trunk Road
Kolkata, West Bengal
India - 700 108

July 2024

Dedicated to

My Parents and my Brother

Declaration of Authorship

I, Debasmita Chakraborty, enrolled in the Ph.D. program within the Applied Statistics Unit at the Indian Statistical Institute, Kolkata, solemnly affirm that the research outlined in this thesis entitled "**Advanced Techniques in Symmetric Key Cryptanalysis**" is entirely my own work. I attest that to the best of my understanding, the contents of this thesis have not been previously published by any other individual, nor have they been utilized in their entirety or in part for the fulfillment of any academic qualification or award.

Debasmita Chakraborty

Debasmita Chakraborty
Applied Statistics Unit
Indian Statistical Unit, Kolkata
203, Barrackpore Trunk Road
Kolkata 700108, INDIA.

Certificate from Supervisor

This is to certify that the work contained in the thesis entitled "**Advanced Techniques in Symmetric Key Cryptanalysis**", submitted by **Debasmita Chakraborty** for the award of the degree of Doctor of Philosophy in Computer Science to Indian Statistical Institute, Kolkata, is a record of the bonafide research works carried out by her under my direct supervison and guidance.

I consider that the thesis has reached the standards and fulfilling the requirements of the rules and regulations relating to the nature of the degree. The contents embodied in the thesis have not been submitted as a whole or as a part for the award of any degree or diploma or any other academic award anywhere given before.

Prof. Mridul Nandi
Professor, Applied Statistics Unit
Indian Statistical Unit, Kolkata
203, Barrackpore Trunk Road
Kolkata 700108, INDIA.

List of Publications

- Debasmita Chakraborty and Mridul Nandi, "*Lower Bound on Number of Compression Calls of a Collision-Resistance Preserving Hash*". IACR Communications in Cryptology, vol. 1, no. 3, Oct 07, 2024. DOI: https://doi.org/10.62056/akgy11zn4.

- Debasmita Chakraborty, Hosein Hadipour, Phuong Hoa Nguyen, and Maria Eichlseder, "*Finding Complete Impossible Differential Attacks on AndRX Ciphers and Efficient Distinguishers for ARX Designs*". IACR Transactions on Symmetric Cryptology, vol. 2024, no. 3, Sept. 2024, pp. 84-176. DOI: https://doi.org/10.46586/tosc.v2024.i3.84-176.

- Debasmita Chakraborty, "*Finding three-subset division property for ciphers with complex linear layers*". Progress in Cryptology - INDOCRYPT 2022. Lecture Notes in Computer Science, vol. 13774, Springer, 2022, pp. 398–421.
DOI: https://doi.org/10.1007/978-3-031-22912-1_18.

- Debasmita Chakraborty and Santu Pal, "*Superpoly recovery of Grain-128AEAD using division property*". Innovative Security Solutions for Information Technology and Communications - 15th International Conference, SecITC 2022. Lecture Notes in Computer Science, vol. 13809, Springer, 2022, pp. 65–80.
DOI: 10.1007/978-3-031-32636-3_4.

Acknowledgments

I am deeply grateful to all the well-wishers whose support has been invaluable in completing this thesis. First and foremost, my heartfelt thanks go to my advisor, Prof. Mridul Nandi, for his unwavering support throughout my PhD journey. He guided me to the field of cryptography and encouraged me to explore its various aspects. His consistent support, insightful comments, and motivating discussions have been the most important factors in building my independent thinking and collaboration on numerous intriguing questions over the years. I am profoundly thankful for his encouragement and for allowing me the freedom to pursue my research passions. It is a privilege to have a supervisor who has constantly supported me in every situation.

During my PhD journey, I had the incredible opportunity to spend six months at TU Graz, Austria, under the guidance of Dr. Maria Eichlseder. Her mentorship significantly transformed my intuition and writing skills, and I am deeply grateful for her academic support and motivation. The memorable experiences, including board game evenings, Christmas parties, delicious pizza treats, and more, have made my time in Graz an unforgettable part of my PhD journey. I also extend my heartfelt thanks to Prof. Christian Rechberger for the enriching academic and non-academic discussions we had during my visit.

My PhD thesis would not have reached completion without the generous support of my collaborators. I express my sincere gratitude to Prof. Mridul Nandi, Dr. Maria Eichlseder, Dr. Santu Pal, Hosein Hadipour, and Phuong Hoa Nguyen for their invaluable contributions. Their willingness to answer my questions and engage in discussions whenever needed was immensely helpful.

I am particularly grateful to Dr. Nilanjan Datta for his unwavering support throughout my research journey, starting from the very first day of my PhD. His positive energy, continuous encouragement, and valuable feedback on my paper drafts were crucial in motivating me to explore various fields in cryptography. I also owe a debt of gratitude to Prof. Bimal Kumar Roy, Prof. Subhamoy Maitra, Dr. Sushmita Ruj, Dr. Avijit Dutta, and Dr. Avik Chakraborty for their insightful comments during my time at ISI. Additionally, I wish to thank my coursework teachers, Prof. Mridul Nandi, Dr. Debrup Chakraborty, Dr. Arijit Bishnu, Dr. Malay Bhattacharya, Dr. Sourav Chakraborty, and Dr. Nilanjan Datta, for their guidance.

I have been fortunate to have had wonderful colleagues and friends throughout my association with ISI. I am lucky to have friends like Manish, Mohammad, Animesh Da, Partha Da, Chandranan, and Abishanka Da. Special thanks to my seniors, Ashwin Da, Bishwajit Da, Soumya Da, Suprita Di, Snehal Di, Arghya Da, Jyotirmoy Da, Debendra Da, Susanta Da, Anik Da, Prabal Da, Nayana Di, and many more who made my stay at ISI memorable.

I apologize to anyone I have not mentioned here, but who contributed to my wonderful experience at ISI. During my stay at TU Graz, I was fortunate to make friends like Hosein, Marcel, Shibam, Lena, Katharina, and Fabian. Special thanks go to my dear friend, Hoa, for her constant support, and cheer throughout my PhD journey. Her presence in Austria during my stay has made my experience truly memorable and enriching. Finally, I extend my heartfelt gratitude to my dear friend Subhra, whose friendship has been a constant source of motivation and encouragement since we first met in 2018. I consider myself incredibly fortunate to have her in my life; her friendship has illuminated every moment, adding a special brightness to my life. Her constant support throughout every step of my PhD journey has been invaluable, and I am deeply grateful to her for it.

I am profoundly thankful to all the non-teaching staff at the Applied Statistics Unit office and the Dean's office for their invaluable support and assistance.

I also want to thank all my teachers and friends from my alma maters: Ramkrishna Sarada Mission Sister Nivedita Girls' School, Kolkata; Jadavpur University, Kolkata; and the Indian Institute of Technology, Kharagpur. This acknowledgment would be incomplete without expressing my gratitude to Prof. Shamik Ghosh, Dr. Buddhadeb Sau, Prof. G. P. Raja Sekhar, Prof. P. V. S. N. Murthy, Prof. Sourav Mukhopadhyay, and Dr. Rupanwita Gayen. I consider myself fortunate to have had such wonderful teachers in my life.

Lastly, I would like to express my deepest gratitude to my family for their unwavering support and encouragement. My parents and brother have been my pillars of strength, providing me with the mental and emotional support needed to navigate this journey. Their continuous encouragement, inspiration, and motivation have been instrumental in giving me the resilience to pursue a career in academia. A Special thanks to my best friend and husband Prajesh, who stands like a pillar through every storm and every calm. Thank you, Ma, Baba, Dada, and Prajesh - for everything.

Debasmita Chakraborty
Kolkata, July 2024

Abstract

Symmetric key cryptographic primitives are essential tools used extensively in daily digital interactions. These primitives are mainly designed to provide three key services: ensuring data confidentiality, maintaining data integrity, and verifying the authenticity of data sources. The primary types of symmetric key primitives that deliver these services include *block ciphers*, *stream ciphers*, *hash functions*, *message authentication codes*, and *authenticated encryption with associated data*. This thesis mainly explores the security analysis of hash functions, several block ciphers, and stream ciphers using some advanced cryptanalytic techniques.

We begin by examining the collision security of a hash function, specifically under the assumption that the underlying compression functions are collision-resistant. This characteristic is termed the collision-resistance preserving property of a hash function. Notably, this property is present in both the Merkle-Damgård and Merkle tree hash structures. This makes us wonder if it is possible to lower the number of calls to the underlying compression function while still maintaining the property. In pursuit of this question, we prove that for an ℓn-to-sn-bit collision-resistance preserving hash function, designed using r tn-to-n-bit compression function calls, it must hold that $r \geqslant \lceil(\ell - s)/(t - 1)\rceil$, assuming all operations other than the compression function are linear. Shifting our focus, we delve into advanced techniques for enhanced cryptanalysis of block and stream ciphers. Initially, we concentrate on the impossible differential (ID) and zero correlation (ZC) attacks, which are pivotal cryptanalytic methods for block ciphers. We introduce an advanced, unified constraint programming (CP) approach based on satisfiability for identifying ID distinguishers in ARX and AndRX ciphers alongside a similar method for pinpointing ZC distinguishers. Furthermore, we extend our novel model to formulate a unified optimization problem that incorporates the distinguisher and key recovery for AndRX designs. Our approach not only enhances ID attacks but also unveils new distinguishers for various ciphers, including SIMON, SPECK, Simeck, ChaCha, Chaskey, LEA, and SipHash. Another significant cryptanalytic technique, particularly applicable to the analysis of block and stream ciphers, is the division property—an advanced version of integral cryptanalysis. Here, we explore the feasibility of the MILP method for the bit-based division property using three subsets (BDPT) propagation in ciphers with complex linear layers. We apply our novel method to discover integral distinguishers based on BDPT for the SIMON, SIMON(102), PRINCE, MANTIS, PRIDE, and KLEIN block ciphers. The integral distinguishers identified by our method are superior to or consistent with the longest existing distinguishers. Finally, we investigate the cube attack, a powerful cryptanalytic technique against stream ciphers. We study the NIST lightweight 3rd round candidate Grain-128AEAD through the lens of division property-based cube attacks.

Initially, we introduce some effective cubes and construct an algorithm to identify conditional key bits for these cubes in Grain-128AEAD. Subsequently, we employ the three-subset division property without unknown subset based cube attacks to recover precise superpolies for Grain-128AEAD in the weak-key setting, yielding improved results.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

- **ANF**: **A**lgebraic **N**ormal **F**orm
- **BDPT**: **B**it-based **D**ivision **P**roperty using **T**hree subsets
- **CBDP**: **C**onventional **B**it-based **D**ivision **P**roperty
- **MILP**: **M**ixed **I**nteger **L**inear **P**rogramming
- **SAT**: **Sat**isfiability
- **SMT**: **S**atisfiability **M**odulo **T**heories
- **LFSR**: **L**inear **F**eedback **S**hift **R**egister
- **NLFSR**: **N**on**L**inear **F**eedback **S**hift **R**egister
- **ID**: **I**mpossible **D**ifferential
- **ZC**: **Z**ero **C**orrelation
- **DDT**: **D**ifference **D**istribution **T**able
- **LAT**: **L**inear **A**pproximation **T**able
- **Sbox**: **S**ubstitution box
- **SPN**: **S**ubstitution **P**ermutation **N**etwork
- **XOR**: Exclusive-**OR**

List of Symbols

- $\Pr[A]$: Probability of occurrence of an event A.
- $\mathbb{N}$: The set of natural numbers $\{1, 2, 3, \ldots\}$.
- $[k]$: The set $\{1, 2, \ldots, k\}$, for any $k \in \mathbb{N}$.
- $[a..b]$: $\{a, a+1, \ldots, b\}$, for any two integers a, and b.
- $a \parallel b$: $(a_0, a_1, \ldots, a_{k-1}, b_0, b_1, \ldots, b_{\ell-1})$.
- $(a_i)_{i\in[k]}$: $(a_1, a_2, \ldots, a_k)$
- a_I: $(a_i \ : \ i \in [k])$, for $I \subseteq [k]$.
- $\mathbb{F}_2$: The finite field $\{0, 1\}$.
- $\mathbb{F}_2^n$: The finite field $\{0, 1\}^n$.
- a_i: i-th component of the vector a.
- For $x, u \in \mathbb{F}_2^n$ x^u: $\prod_{i=0}^{n-1} x_i^{u_i}$.
- $a \geq b$: $a_i \geqslant b_i$, for all $i = 0, 1, \ldots, (n-1)$.
- $a > b$: $a_i > b_i$, for all $i = 0, 1, \ldots, (n-1)$.
- $a \not\geq b$: There exists at least one $i \in \{0, 1, \ldots, n-1\}$ such that, $a_i < b_i$.
- $a{\restriction_I}$: $(a_0, a_1, \ldots, a_{n-1})$ satisfying $a_i = 1$, if $i \in I$, and $a_i = 0$, otherwise.
- $\mathcal{S} \leftarrow a$: $\mathcal{S} = \mathcal{S} \cup \{a\}$, where $\mathcal{S} \subseteq \mathbb{F}_2^n$.
- $\mathcal{S} \rightarrow a$: $\mathcal{S} = \mathcal{S} \backslash \{a\}$, where $\mathcal{S} \subseteq \mathbb{F}_2^n$.
- $|\mathcal{S}|$: The cardinality of the set $\mathcal{S}$.
- e_i: The i-th unit vector in $\mathbb{F}_2^n$.
- $\mathbf{0} = 0^n$, $\mathbf{1} = 1^n$.
- $wt(a)$: The hamming weight of a vector a

- $a \lll i$: The vector a is shifted left by i bits.
- $a \ll i$: The vector a is rotated left by i bits.
- $A^{\mathbf{T}}$: Transpose of the matrix A (also used with vectors).
- $\mathsf{span}(\mathcal{S})$: The subspace spanned by a set of vectors $\mathcal{S}$.
- $\mathsf{rank}(\mathcal{S})$: The maximum number of linearly independent vectors in the set $\mathcal{S}$.

Symbols that are not listed are either standard or only used once.

Chapter 1

Introduction

Cryptography, a term originating from the combination of the Greek words *kryptos*, which translates to "hidden secret", and *graphein*, meaning "to write", refers to the field dedicated to secure communication when faced with potentially harmful individuals. Cryptography is strongly associated with contemporary electronic communication, but its roots can be traced back to ancient times. Examples of early cryptography can be found around 2000 B.C., during the era of ancient Egypt, where non-standard hieroglyphics were employed as a means of secretive communication. Additional notable instances of cryptography throughout history include the Caesar and Vigenere ciphers and the German Enigma machine used during World War II. While these earlier encryption methods were considered effective at the time of their use, they were eventually broken through mathematical analysis, cryptanalytic techniques, and human ingenuity, well before advances in modern computing power made cryptanalysis significantly more efficient.

Cryptography has undergone significant evolution in contemporary society, emerging as a distinct field at the crossroads of mathematics and computer science. Upon examination, it becomes evident that the broader term is *cryptology* rather than *cryptography*. Cryptology encompasses two primary branches:

- Cryptography refers to the scientific practice of concealing the intended message's meaning through secret writing.
- Cryptanalysis involves the scientific and sometimes artistic pursuit of deciphering cryptosystems. As cryptanalysis is crucial for verifying the security of a cryptosystem, it plays an essential role within cryptology.

In the present era, cryptography has transformed and evolved into what we now refer to as modern cryptography. This discipline draws heavily from intricate mathematical theories, including but not limited to integer factorization, discrete logarithms, and the utilization of one-way functions. For example, recovering a secret key hidden in a large search space with exponential complexity is essential to cryptography. In this exposition, we aim to give a thorough overview of the main issues in modern cryptography and explore its key areas.

1.1 Modern Cryptography

Imagine a scenario where `Alice` and `Bob` possess a confidential key, denoted as k, and `Alice` intends to send a message, represented as m, to `Bob` through a network. However, the primary concern is to ensure the confidentiality of m, especially in the face of potential interception by an adversary, referred to as `Eve`. In the scenario where `Alice` transmits the message m to `Bob`, it leaves the message vulnerable to interception by the adversary, `Eve`, who can easily access the content of the plaintext. `Eve` could exploit this vulnerability by altering m into m' and forwarding it to `Bob`. Notably, `Bob` remains oblivious to this modification, making it impossible for him to discern whether the message originated from `Alice` or `Eve`. Moreover, neither `Alice` nor `Bob` can refute the authenticity of the received message. Hence, the primary challenge within modern cryptography revolves around mitigating these vulnerabilities to ensure secure communication across inherently insecure channels. Achieving secure communication requires focusing on four distinct objectives.

- Confidentiality safeguards information, ensuring only authorized individuals can access it. Secrecy and privacy are often used interchangeably with confidentiality, denoting protecting sensitive data. For example, only the sender (`Alice`) and the intended recipient (`Bob`) should have access to the content of the message m, ensuring complete confidentiality during transmission. Various methods exist to maintain confidentiality, ranging from physical security measures to using complex mathematical algorithms that encode data, making it incomprehensible to unauthorized parties.

- Data Integrity is a critical service that focuses on preventing unauthorized modifications to data. Preserving data integrity involves the ability not only to detect but also to respond to any unauthorized alterations made by parties without proper authorization. These alterations can encompass a range of actions, including inserting new data, removing existing data, or substituting legitimate data with false or misleading information. For example, preventing an active adversary, who alters the message from m to m', from having m' accepted is essential. Put differently, any modifications made to the exchanged messages should be readily detectable by either `Alice` or `Bob`, ensuring the integrity of the communication. Maintaining data integrity is essential for ensuring data reliability, accuracy, and trustworthiness within systems and databases.

- Authentication is a service closely linked to the identification process. It encompasses both entities involved in a transactions and the information being exchanged. When two parties engage in communication, they must establish each other's identities securely. Specifically, `Alice` and `Bob` need to discern whether the exchanged messages originate from themselves or an adversary, ensuring the security of their communication. Moreover, any information transmitted over a channel must undergo authentication procedures to verify its source, the time of origin, the content, and the time it was sent. To address these requirements, cryptography often divides authentication into two main categories: entity authentication and data origin authentication. Notably, data origin authentication inherently ensures data integrity, as any modification to a message would change its source.

- NON-REPUDIATION is a crucial service that prevents an entity from disavowing prior commitments or actions. When conflicts arise from an entity refusing to acknowledge specific past actions, it becomes essential to have mechanisms to address and resolve such disputes. For instance, a scenario might involve one entity authorizing another to purchase property, only for the authorizing entity to later deny granting such authorization. In the scenarios discussed above, we've assumed `Alice` and `Bob` are trustworthy, with `Eve` as the adversary. However, if `Alice` or `Bob` were to become adversarial, it's critical that any denial of their commitments regarding the exchanged messages can be readily identified. This ensures accountability and integrity within their communications. In such cases, a formal procedure involving the intervention of a trusted third party is necessary to facilitate resolution and establish the authenticity of the commitments or actions in question.

A primary objective of cryptography is to manage the aforementioned four security services, both theoretically and practically, effectively. Cryptography is primarily categorized into *symmetric cryptography* and *asymmetric cryptography*, depending on the secret key distribution among the parties involved. A third category, often referred to as *cryptographic protocols*, utilizes both cryptographic methods as foundational elements. Below, we provide a detailed explanation of each category.

1.1.1 Symmetric Cryptography

Symmetric cryptography involves two parties using a shared secret key to encrypt and decrypt messages. Historically, from ancient times until 1976, all cryptographic methods were symmetric. Symmetric ciphers continue to be widely utilized, particularly for encrypting data and verifying the integrity of messages. Symmetric cryptography can be best understood through a simple scenario: Imagine two individuals, `Alice` and `Bob`, who wish to exchange information over a channel that is not secure. The term channel broadly refers to any medium through which communication occurs, such as the Internet, the air used in mobile or wireless LAN communications, or any other conceivable medium. The core issue arises when an unauthorized individual, `Eve`, gains access to this channel, a practice known as eavesdropping. In many cases, `Alice` and `Bob` would prefer their conversations to remain private, away from `Eve` 's prying ears.

In this context, symmetric cryptography serves as an effective strategy. `Alice` sends her message m, encrypting it into ciphertext c with the help of the key k using a symmetric encryption method. `Bob` then receives this ciphertext and decrypts it using the same key k, effectively reversing the encryption process.

What is the benefit?

With a robust encryption algorithm, the ciphertext appears as nothing more than random bits to `Eve`, essentially providing no valuable information to her. The system requires a secure method for distributing the key between `Alice` and `Bob`. In this scenario, we address the issue of confidentiality, which involves concealing the message contents from an eavesdropper. However, cryptography offers a range of other capabilities, including safeguarding

against unnoticed alterations to the message by `Eve` (*message integrity*) and ensuring that the message indeed originates from `Alice` (*sender authentication*).

Symmetric key algorithms encompass various types, including *stream ciphers*, *block ciphers*, *hash function*, *message authentication codes (MAC)*, and *authenticated encryption (AE)*. Stream ciphers and block ciphers serve the purpose of ensuring data confidentiality. Typically, block ciphers employ the secret key to encrypt a plaintext block into ciphertext. In contrast, stream ciphers encrypt the plaintext through `XOR` operations with a keystream generated using the secret key. A cryptographic hash is an unkeyed function that accepts messages of any length as input and produces a concise fixed-length *digest* of the message as output. MAC ensures data integrity by creating a tag on the sender's side, which is verified on the receiver's end. Encryption modes are employed to encrypt messages of variable lengths using block ciphers. AE algorithms offer both data confidentiality and integrity by encrypting messages of any length and generating tags. With the increasing use of cryptography in devices with limited resources, lightweight cryptography has gained prominence.

1.1.2 Asymmetric Cryptography

Symmetric cryptography has a long history, having been in use for over 4000 years. In contrast, asymmetric cryptography is a relatively recent development. It was first introduced to the public in 1976 by Whitfield Diffie, Martin Hellman, and Ralph Merkle. Asymmetric cryptography was developed to overcome the limitations of symmetric cryptography, particularly the need for a shared secret key between parties. In symmetric cryptography, both the sender and the receiver must use the same key for encryption and decryption, which requires securely exchanging the key beforehand—a challenging task, especially over insecure communication channels. Even after addressing the key distribution challenge, managing a potentially vast number of keys remains an issue. To address these limitations, Diffie, Hellman, and Merkle introduced a groundbreaking concept: the key used by the sender to encrypt the message does not need to remain secret.

More precisely, asymmetric cryptography differs from symmetric cryptography by introducing the concept of both a secret key, similar to symmetric cryptography, and a public key. As an illustration, when `Alice` intends to transmit a confidential message to `Bob` using RSA encryption, she employs Bob's public key for encryption. Consequently, only Bob, possessing his private key, can decrypt and access the message. In addition to enabling secure communication, asymmetric cryptography offers significant benefits, such as the ability to create digital signatures. Digital signatures verify the authenticity and integrity of documents or messages, ensuring that they were indeed sent by the claimed sender and have not been tampered with.

The security of asymmetric cryptography relies on computational problems that are hard to solve without the private key, such as factoring large numbers (in RSA) or solving discrete logarithms (in Diffie-Hellman and Elliptic Curve Cryptography). By addressing these challenges, asymmetric cryptography has become the foundation for secure internet communications, including HTTPS, email encryption, and blockchain technologies.

1.1.3 Cryptographic Protocols

A cryptographic protocol is a set of guidelines that relies on symmetric and asymmetric techniques to ensure secure communication. An example is the Transport Layer Security (TLS) protocol, which is used in all modern web browsers. When you visit a secure website (such as those with "https" in the URL), TLS activates. First, it uses asymmetric methods to establish a secure connection and exchange keys. Then, it switches to symmetric cryptography, using the established keys to encrypt and authenticate the messages being sent back and forth between your browser and the website's server. This ensures the data stays safe from prying eyes while it travels over the internet.

1.1.4 Cryptanalysis

Cryptanalysis plays a crucial role in evaluating the security of cryptographic systems, as it helps identifying weaknesses that could be exploited by attackers. According to the Kerckhoffs' Principle,

"A cryptosystem should be secure even if the attacker knows all details about the system, with the exception of the secret key. In particular, the system should be secure when the attacker knows the encryption and decryption algorithms."

So, cryptanalysis involves techniques and methods to decipher encrypted messages without knowledge of the key. Cryptanalysts typically aim to find weaknesses or vulnerabilities in cryptographic algorithms, protocols, or implementations to exploit them and gain unauthorized access to encrypted data. In addition, cryptographers use the insights from cryptanalysis to design more robust and resilient cryptographic algorithms and protocols. Cryptanalysis is divided into two primary categories: mathematical cryptanalysis and implementation attacks. The first focuses on the mathematical foundation of the cryptographic system, while the second utilizes side-channel data, such as power usage or timing, to break into a cryptosystem. For both types of attacks, an attacker needs specific resources—data, time, and memory storage, which are referred to as metrics of attack complexity. The efficiency of these complexity metrics enhances the effectiveness of an attack. As detailed in the following section, various methods can be used to perform cryptanalysis on cryptographic systems.

1.1.5 Tools for Cryptanalysis

As a result of Kerchkoffs' principle, a cryptographic algorithm must be constructed in such a manner that even if an attacker possesses complete knowledge of the scheme's structure, they cannot compromise its security without access to the key. Therefore, developing symmetric primitives and schemes is a challenging and frequently time-consuming endeavour, necessitating significant cryptanalysis efforts. The rise of cryptographic competitions has become a popular approach for cultivating concentrated efforts, pinpointing the most prospective candidates, and improving overall confidence in their security. United States National Institute of Standards and Technology (NIST) or ECRYPT, a European Union-funded Network of Excellence, initiate public competitions for specific types of cryptographic schemes. The

global cryptographic research community can suggest candidate ciphers and collaboratively analyze them over several years.

Numerous cryptographic algorithms undergo standardization processes, such as the Authenticated Encryption Standard (AES) [1], [2], Secure Hash Algorithm 3 (SHA3) [3], the CAESAR competition [4], the New European Schemes for Signatures, Integrity, and Encryption (NESSIE) [5] project, eStream [6], the Lightweight Cryptography (LWC) [7] competition and others. Algorithms submitted to these competitions are publicly disclosed and subjected to a thorough analysis over several years. Following public scrutiny, a single algorithm or a suite of algorithms is standardized, depending on the competition's criteria. Therefore, creating cryptographic schemes alone does not fulfil the requirements for establishing a secure system. A thorough analysis of these systems is essential to instil confidence in their reliability and suitability for practical applications.

The effectiveness of these competitions relies significantly on the comprehensive evaluation of all participants. This task becomes increasingly challenging as the number of contenders continues to rise. The complicated structure of certain candidates and design methodologies that are less amenable to manual cryptanalysis have led to a significant reliance on automated cryptanalysis tools. In addition, performing cryptanalysis on various cryptographic schemes can be time-consuming and potentially error-prone when approached manually. Furthermore, cryptanalysis often involves computationally intensive tasks, such as searching for vulnerabilities or optimizing parameters. Automated tools leverage computational power to tackle these tasks more effectively than manual methods by framing the procedure as search or optimization problems, leading to faster discoveries and advancements in cryptographic techniques. Such challenges are often tackled using existing tools like satisfiability (SAT), mixed-integer linear programming (MILP), and constraint programming (CP) solvers, or specialized tools integrating techniques from related fields. Additionally, automated tools provide consistency and reproducibility in cryptanalysis. Results obtained through automated processes can be easily verified and replicated by other researchers, enhancing the reliability of findings and facilitating collaboration within the cryptographic community. Widely utilized applications include techniques such as differential cryptanalysis, linear cryptanalysis, the division property, and others. In summary, these tools for cryptanalysis are indispensable for advancing cryptographic research, improving security, and ensuring the robustness of cryptographic systems in an increasingly digital world.

1.2 Thesis Outline and Contribution

The thesis predominantly focuses on analyzing the security of symmetric-key cryptographic primitives through advanced cryptanalytic techniques. While the primary goal is to develop and apply new cryptanalysis methods, a complementary aspect explored in Chapter 3 involves a provable security analysis of hash modes. The interplay between provable security and cryptanalysis is fundamental in modern cryptography. Cryptanalysis aims to identify weaknesses in cryptographic designs by actively searching for attacks, whereas provable security seeks to establish formal security guarantees under well-defined theoretical assumptions. While these two approaches may seem distinct, they are inherently interconnected: cryptanalysis provides empirical validation by testing whether a cryptographic scheme holds up

against practical attacks, while provable security offers a rigorous framework to reason about security under idealized models. Together, they create a more comprehensive understanding of cryptographic strength—provable security sets necessary conditions for security, and cryptanalysis assesses whether these conditions hold in real-world scenarios.

In particular, the investigation of collision-resistance preserving properties in Chapter 3 aligns with the overarching theme of assessing symmetric-key security. By proving lower bounds on the number of compression function calls necessary for preserving collision resistance, this work contributes to understanding fundamental limits in hash function design. This analysis, though distinct from direct attack methodologies, ultimately serves the same goal as cryptanalysis: evaluating and strengthening the security of symmetric-key schemes.

On the other hand, this thesis also explores the cryptanalysis of various symmetric-key primitives, including block ciphers and stream ciphers, using classical techniques. Additionally, it introduces novel automated methods to enhance the efficiency of cryptanalysis across a wide range of symmetric-key cryptosystems. Thus, the study introduces sophisticated techniques—ranging from automated attack frameworks to theoretical security bounds—all aimed at deepening our understanding of symmetric-key cryptographic security. The primary research of the thesis is organized as follows.

- In Chapter 2, we commence with a concise overview of symmetric cryptographic primitives, particularly emphasizing block ciphers, stream ciphers, and hash functions. Subsequently, within this chapter's latter section, we provide a brief exposition of fundamental cryptanalytic methodologies, which will be further explored in subsequent thesis chapters. Furthermore, we briefly describe all symmetric cryptosystems, including block ciphers and stream ciphers, subjected to cryptanalysis within this thesis.

- In Chapter 3, we put our focus on the collision security of a hash function, specifically under the assumption that the underlying compression functions are collision-resistant. This property of a hash mode is called *collision-resistance preserving property*. Notably, both the Merkle-Damgård and Merkle tree hash structures exhibit this property, prompting the question of whether it is possible to reduce the number of underlying compression function calls while maintaining the collision-resistance preserving property.

  While Merkle-Damgård and Merkle tree hash mode can process ℓ block messages by making ℓ or $(\ell - 1)$ calls to the underlying $2n$-to-n bit compression function, in [8], [9] two important hash modes T_5, and ABR have been proposed which can handle an extra message block than Merkle-Damgård and Merkle tree while maintaining same count of compression function calls. Moreover, in [8], [10], the collision security of T_5, and ABR hash mode are discussed under the ideal model assumptions of the compression functions. However, it is an open problem to establish the collision-resistance preserving property of these hash modes.

  In pursuit of this question, first, we prove that T_5 and ABR hash mode do not possess the collision-resistance preserving property. Additionally, we provide a lower bound on the compression function calls of a collision-resistance preserving hash function, assuming all the operations other than the compression functions are linear in the hash mode. Precisely, we prove that, for an ℓn-to-sn-bit collision-preserving hash

function, designed using r tn-to-n-bit compression function calls, it must hold that $r \geqslant \lceil (\ell - s)/(t-1) \rceil$. This result shows that Merkle-Damgård and Merkle tree hash are indeed optimum constructions in the light of collision-resistance preserving property. This work is published in [11].

- Chapter 4 focuses on the essential cryptanalytic strategies of impossible differential attack (ID attack) and zero-correlation attack (ZC attack), which are fundamental for evaluating block ciphers. To thoroughly investigate impossible differential (ID) and zero-correlation (ZC) attacks on block ciphers, one must go through two key phases: firstly, identifying a distinguisher, and secondly, recovering the key using that distinguisher. This approach requires meticulous tracking of how differences (or linear masks) propagate at the word or bit level within the cipher. Attempting this manually can be extremely time-consuming and is prone to errors. Notably, in [12], [13], the authors introduced both word-based and bit-oriented CP-based models for ID/ZC attacks based on the satisfiability of the CP model, extendable to a unified constraint optimization model for key recovery, applicable to strongly/moderately aligned designs like SKINNY, and weakly aligned designs such as Ascon. While the methods in [12], [13] improved the best-known ID/ZC attacks on several SPN ciphers, their application to an essential category of block ciphers, i.e., Addition-Rotation-XOR (ARX) and And-Rotation-XOR (AndRX) designs, was left for future work.

  In this chapter, we extend the methods proposed at EUROCRYPT 2023 and ToSC 2024 for finding ID attacks [12], [13] from different aspects. First, we provide a CP-based model based on satisfiability to find ID distinguishers for ARX and AndRX ciphers. We also show the applicability of our new CP models for finding ZC distinguishers. Next, we show how to extend the CP-model for key recovery in [12], [13] for bit-wise designs, particularly for AndRX designs. Lastly, we put our new models for distinguisher and key-recovery parts into a unified CP model for finding the complete ID attacks, including the key recovery evaluations.

  To show the usefulness of our methods, we apply them to find ID distinguishers/attacks on several ARX (SPECK [14], ChaCha [15], Chaskey [16], SipHash [17], and LEA [18]) and AndRX (SIMON [14], and Simeck [19]) ciphers and improve the best previous results. We provide ID distinguishers for ChaCha, Siphash, SPECK-96, and SPECK-128 for the first time, along with that, we provide several new ID distinguishers for Chaskey and SPECK with truncated input/output differences. Additionally, we improve ID attacks on SIMON-64-96, SIMON-64-128, SIMON-128-128, and SIMON-128-256 by one round, and SIMON-128-192 by two rounds. This work is published in [20].

- In Chapter 5, we concentrate on another significant cryptanalytic technique, particularly applicable to the analysis of block ciphers and stream ciphers, which is the division property- an advanced version of integral cryptanalysis.

  In [21], one of the most advanced variants of the division property, known as the bit-based division property using three subsets (BDPT), was introduced. Later, in [22], the authors proposed the variant three subset division property (VTDP), which they used to enhance some integral distinguishers, though it did so at the expense of some accuracy

of BDPT. To address this, in [23], the authors introduced a method for modeling the propagation of BDPT. More recently, in [24], the authors proposed a model set method for searching integral distinguishers based on BDPT. These approaches have been applied to block ciphers that feature a simple bit permutation as their linear layer.

To broaden the applicability of BDPT, we introduce a method to accurately determine BDPT propagation through complex linear layers. We then develop an advanced automatic search algorithm for BDPT to identify integral distinguishers for block ciphers with complex linear layers. For the first time, we apply BDPT to block ciphers with complex linear layers, enhancing integral distinguishers for PRINCE [25], MANTIS [26], KLEIN [27], and PRIDE [28] by one round. Additionally, our model is applied to all variants of SIMON, including SIMON(102) block ciphers. The distinguishers we discover are consistent with the previously longest distinguishers [24], but our tool operates faster than previous tools [24], [29]. This work was published at [30].

- In Chapter 6, we examine the cube attack, a potent cryptanalytic method introduced by Dinur and Shamir in [31], known for its effectiveness against symmetric cryptosystems.

  Given that Grain-128AEAD [32] is a finalist of the NIST lightweight cryptography competition, its cryptanalysis is a significant area of research. By applying division property-based cube attacks, exact superpolies for the 190, 191, 192-round versions of Grain-128AEAD have been effectively recovered, enabling key-recovery attacks [33]–[35]. However, these cube attacks involve relatively high cube dimensions.

  In this context, we initially search for efficient cubes with lower dimensions than previous division property-based cube attacks, aiming to recover superpolies effectively. We introduce several cubes with dimensions 91, 92, 93, and 94 for Grain-128AEAD. We propose an algorithm that sets specific conditions on key bits based on cube variables. To apply our method, we utilize the three-subset division property without unknown subsets to accurately recover the superpoly of Grain-128AEAD in a weak-key setting. We successfully determine exact superpolies for 192-195 rounds of Grain-128AEAD in this setting, marking the best results for Grain-128AEAD in the weak key setup best known to us. This work was published in [36].

- Chapter 7 provides a summary of all the findings presented in this thesis and discusses potential avenues for future research.

Chapter 2

Preliminaries

2.1 Symmetric Cryptographic Primitives

Symmetric key cryptographic primitives are essential tools used extensively in daily digital interactions. They play a crucial role in ensuring secure communications across various platforms, including TLS (Transport Layer Security) versions 1.2 and 1.3, IPSec (Internet Protocol Security), SSL (Secure Socket Layer), as well as in numerous other applications such as online and chip-based payment systems, wireless networks, and RFID (Radio Frequency Identification) technology. These primitives are primarily designed to provide three essential services: ensuring data confidentiality, maintaining data integrity, and verifying the authenticity of data sources. The primary types of symmetric fundamental primitives that deliver these services include *stream ciphers*, *block ciphers*, *hash functions*, *message authentication codes*, and *authenticated encryption with associated data (AEAD)*. Below, we provide a broad overview of block ciphers, stream ciphers, and hash functions relevant to this thesis.

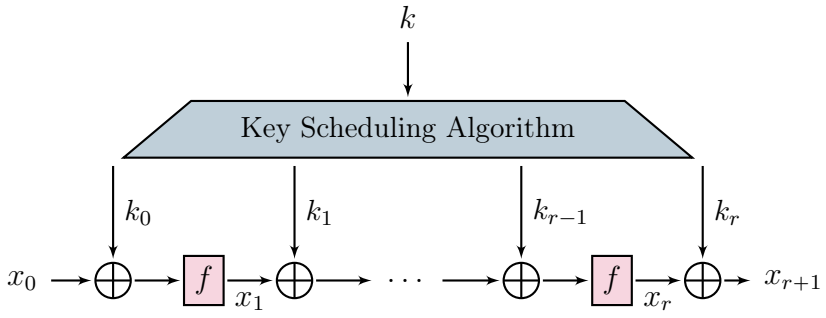


Figure 2.1: Iterated construction for a block cipher

2.1.1 Block Ciphers

A block cipher consists of a deterministic algorithm, denoted as $\mathcal{E} = (E, D)$, which manages the encryption and decryption processes of a block of bits. Here, E represents the encryption algorithm, while D represents the decryption algorithm. If we consider the key space of $\mathcal{E}$ as $\mathcal{K}$ and the message space as $\mathcal{X}$, then $\mathcal{E} = (E, D)$ is a block cipher designed for use over

the domain $(\mathcal{K}, \mathcal{X})$.

The encryption algorithm, denoted as E, operates as a function $E : \mathcal{K} \times \mathcal{X} \to \mathcal{X}$. This means that for each key $k \in \mathcal{K}$, the function $E_k : \mathcal{X} \to \mathcal{X}$ yields a permutation. Symmetrically, the decryption algorithm, D, functions as the inverse permutation of E_k, ensuring that $D_k(E_k(x)) = x$ for all inputs $x \in \mathcal{X}$ and keys $k \in \mathcal{K}$. This condition of correctness ensures that after encryption followed by decryption, the original message is obtained, maintaining the integrity of the data.

Generic Structures of Block Ciphers

A prevalent approach to creating a block cipher involves the iterative modification of an internal state initially set by the plaintext. This process entails the repeated application of a fundamental transformation known as the round function to the internal state. Concurrently, a key schedule algorithm is implemented to generate subkeys derived from the original secret key. These subkeys are incorporated into the internal state, ensuring the key influences the round functions. Having examined the encryption and decryption processes utilizing a block cipher, we then discuss the structural specifics of the encryption and decryption algorithm. One common approach to designing the round function involves the *Feistel structure* or the *Substitution Permutation Network (SPN)*. These configurations determine the specific operations executed in each block cipher round, shaping the encryption and decryption processes.

SPN round function. In the SPN round function, each round processes all bits simultaneously, incorporating both a linear and a non-linear layer. The non-linear layer generally includes multiple small permutations, known as S-boxes, which work concurrently on different segments of the internal state. On the other hand, the linear layer involves a linear operation that blends the S-box outputs, typically through matrix multiplication. The subkey is combined with the internal state using an XOR operation.

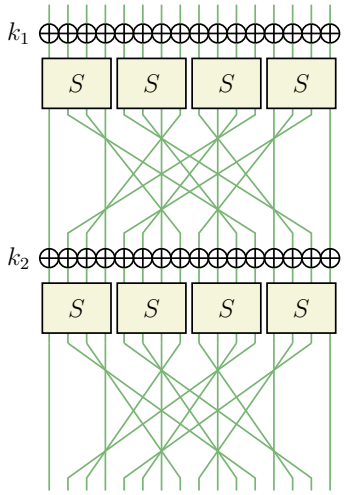


Figure 2.2: Block Cipher Utilizing Substitution-Permutation Network (SPN) Round Function

A prominent instance of an SPN round function is observed in the AES [2] (Advanced Encryption Standard), where the block size is 128 bits and the S-boxes operate on 8-bit segments. The linear transformation in AES consists of the ShiftRows and MixColumns operations. Additionally, lightweight block ciphers like PRESENT [37] and GIFT [38] also utilize SPN structures but with smaller parameters: block size of 64 bits and S-boxes operate on 4 bits, respectively, where the linear layer is implemented as a bit permutation. These examples highlight the adaptability of SPN designs to different security requirements and hardware constraints, making them suitable for various encryption tasks.

Feistel round function. In a Feistel round function, only one-half of the total bits are processed during each round. This structure divides the block into two halves; during each round, the operation applies transformations to one half while the other remains unchanged, allowing efficient and effective data mixing. Here, the round function uses a keyed function F_{k_j} and splits the internal state into two branches L_j and R_j. It can be defined as:

$$\begin{cases} L_{j+1} = R_j \\ R_{j+1} = L_j \oplus F_{k_j}(R_j) \end{cases}$$

This type of round function is simple to reverse, which enables both the encryption and decryption processes to utilize the function F_{K_j} in the same direction, negating the need for F_{K_j} to be a permutation. The former NIST standard, the Data Encryption Standard (DES) [39], is the most well-known block cipher that employs this architecture. Other important and famous examples are SIMON [14] and Simeck.

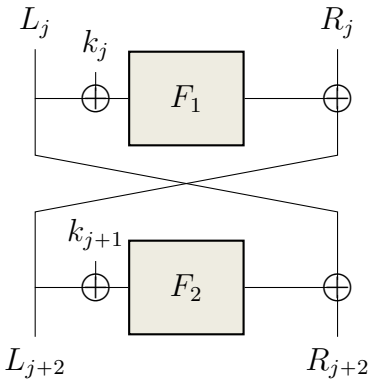


Figure 2.3: 2-round Feistel structure

2.1.2 Stream Ciphers

A stream cipher is an encryption algorithm that encrypts data one bit or byte at a time. Unlike block ciphers, which encrypt data in fixed-size blocks (typically 64 or 128 bits), stream ciphers work with continuous data streams, making them suitable for environments where data arrives unpredictably or where it is impractical to handle data in large blocks. From a mathematical perspective, a stream cipher is described as a function F that, upon receiving a secret key k and an initialization vector (IV), produces a sequence of keystream bits, denoted $z_0, z_1, \ldots$ like this. The encryption and decryption are constructed using bitwise

XOR operation such as $y_i = x_i \oplus z_i$, and $x_i = y_i \oplus z_i$, respectively. In this context, it is assumed that with a constant key k, the initialization vector IV should not be reused with the same key to prevent the repetition of the keystream.

The security of the keystream largely hinges on the selection of F. Firstly, the keystream should appear as random as possible to an observer. Secondly, even if they have some keystream bits, they shouldn't be able to predict the next bit with any significant accuracy.

The function F typically operates in two steps. In the first step, named *key initialization step*, we prepare the system with the key and initialization vector. Therefore, we go through a setup process without outputting any bits. Finally, during the *keystream generation step*, we produce the keystream bit by bit while updating the system. We keep doing this until we have enough keystream bits.

Examples. One of the most famous stream ciphers, RC4, has historically been used in protocols like WEP (Wired Equivalent Privacy) and TLS (Transport Layer Security). However, it has fallen out of favour due to security vulnerabilities. Another important example is ChaCha20, a modern stream cipher that provides high-speed encryption and has been adopted in various security protocols, including TLS and disk encryption, in some systems. Other notable instances of F encompass systems like Grain, Trivium, Lizard, Acorn, and others. These are founded on Non-Linear Feedback Shift Registers (NLFSRs) or a blend of Linear Feedback Shift Registers (LFSRs) and NLFSRs coupled with a filtering function. Stream ciphers are designed to be fast and efficient, especially in constrained environments. However, they require careful implementation and management to ensure security, particularly in generating and handling the keystream.

2.1.3 Hash Functions

Cryptographic hash functions are widely used in cryptography and mainly used in various protocols. They generate a digest of a message, which is a concise, fixed-length string of bits. Its digest or hash value acts as a fingerprint for any message, uniquely characterizing it. In cryptography, hash functions have multiple applications: They are a fundamental component of digital signature schemes and message authentication codes.

Security Requirements of Hash Functions

In contrast to other cryptographic algorithms, hash functions lack the concept of keys. The inquiry now revolves around identifying the attributes necessary for a hash function to ensure security. Therefore, a hash function H should satisfy the following three properties to be secure:

- **Preimage Resistance:** Finding an input message x that satisfies $z = H(x)$, where H denotes a hash function, should be computationally impractical when provided with a hash output z.
- **Second Preimage Resistance:** For a given message x_1, generating a distinct message x_2 with $x_1 \neq x_2$, but with equal hash values $H(x_1) = H(x_2)$, is computationally difficult.

- **Collision Resistance:** A hash function is called *collision resistant* when it becomes computationally challenging to locate two distinct inputs, represented as x_1 and x_2 ($x_1 \neq x_2$), such that their hash values, $H(x_1)$ and $H(x_2)$, coincide.

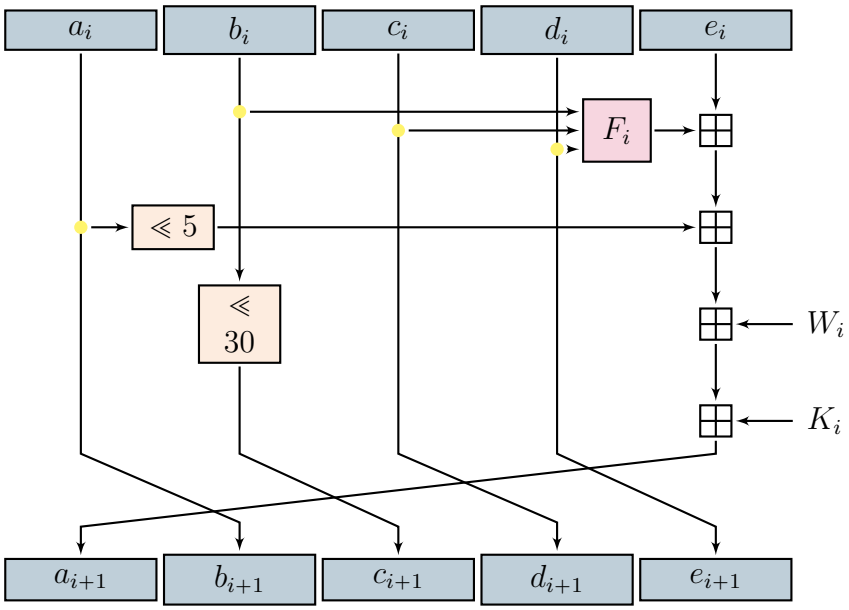


Figure 2.4: SHA-1 step function

Overview of Hash Algorithms

Cryptographic hash functions are fundamental to modern cryptography, providing essential functionality in ensuring data integrity, authentication, and digital signatures. Their design has evolved over time, driven by the need to address emerging security challenges, efficiency requirements, and the increasing complexity of cryptographic applications.

The primary purpose of hash functions is to transform variable-length messages into fixed-length outputs, or digests, which act as unique fingerprints of the original data. This enables the detection of even the smallest modifications to a message. Hash functions are indispensable in digital signature schemes, message authentication codes, and many cryptographic protocols due to their efficiency and reliability.

However, designing secure hash functions is challenging. The earliest hash functions, such as MD5 and SHA-1, were designed using the Merkle–Damgård construction, which relies on iterative processing of fixed-size input blocks with a compression function. These designs were effective for many applications but eventually became vulnerable to practical collision attacks as cryptanalysis techniques improved. This vulnerability underscored the need for more secure and flexible designs. To address these shortcomings, alternative approaches were explored:

- **Dedicated Hash Functions.** These algorithms are purposefully designed and optimized to fulfill the role of hash functions, ensuring they meet the necessary requirements for securely converting variable-length inputs into fixed-length outputs.

- **Block Cipher-based Hash Functions.** In scenarios where dedicated hash functions were unavailable or unsuitable, block ciphers like AES were adapted to construct hash functions. Techniques such as the Matyas-Meyer-Oseas and Davies-Meyer constructions offered a way to leverage existing cryptographic primitives, ensuring efficiency and reusability in resource-constrained environments.
- **Sponge-based Hash Functions.** The sponge construction introduced a paradigm shift, offering a unified framework for both hashing and authenticated encryption. By absorbing input data into an internal state and squeezing out the desired output, this design addressed many limitations of earlier constructions. SHA-3, the most prominent sponge-based hash function, provides enhanced flexibility and strong security properties, making it suitable for diverse applications.

Each construction reflects a response to specific challenges or limitations, illustrating the continuous evolution of hash function design. A detailed explanations can be found as below.

Dedicated Hash Functions. Over the past twenty years, numerous constructions within this category have emerged. In practice, the SHA family of hash functions, including variants like SHA-256 and SHA-3, is commonly used across various cryptographic applications. However, older hash functions such as MD5 and RIPEMD, are largely obsolete and rarely used today.

Hash Functions from Block Ciphers. Block cipher chaining techniques offer another avenue for constructing hash functions. Among the famous constructions arising from block ciphers is the `Matyas-Meyer-Oseas` scheme. To construct this hash function, we start by splitting the message x into fixed-size blocks x_i. Each block x_i is then encrypted using a block cipher E, with the key being the previous chaining value H_{i-1}. Suppose the encryption of the plaintext block x_i is y_i. Therefore, to generate a new chaining value H_i, y_i is `XOR`ed with the plaintext block x_i.

$$H_i = E_{g(H_i)}(x_i) \oplus x_i$$

Two other well-known block cipher-based hash function designs are the *Davies-Meyer* and *Miyaguchi-Preneel* constructions.

Sponge-based Hash Functions. The operation of a sponge-based hash relies on a *sponge construction*, which absorbs input data into a fixed-size state and then squeezes out the output data. The function starts with an initial empty state, divided into the rate (r) and the capacity (c). To ensure the input data fits the rate section properly, the input message is padded, and the padded message is absorbed in blocks the size of the rate. This phase is called *absorbing phase*. Once all input data is absorbed, the hash value is produced by outputting data from the rate section in blocks. This squeezing continues until enough data has been output to meet the desired hash length. This method is distinguished by its ability to *absorb* input data and then *squeeze* out the desired output, making it highly effective and secure for cryptographic purposes.

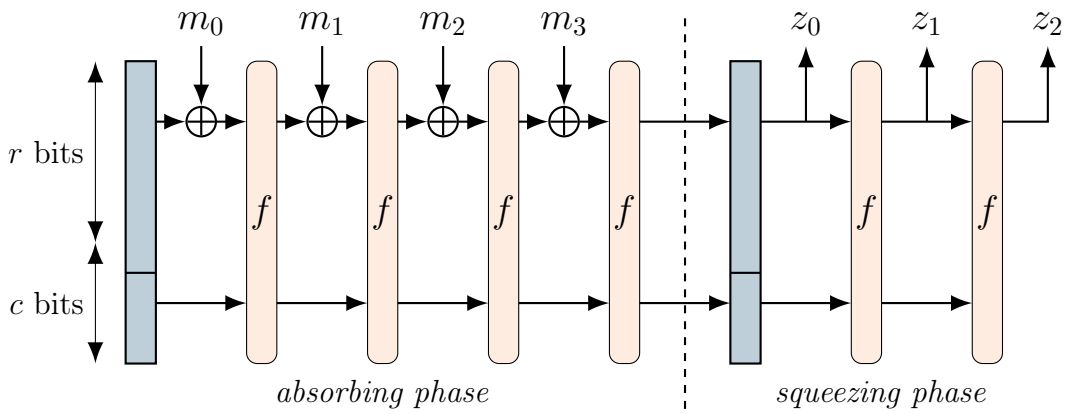


Figure 2.5: Sponge construction

2.2 Basic Cryptanalytic Techniques

Classical cryptanalysis typically employs algorithms to recover either the plaintext m from the encrypted message c or to deduce the key k used in encryption from the encrypted message c. As previously discussed, cryptanalysis is generally categorized into two main types: mathematical cryptanalysis and implementation attacks. In mathematical cryptanalysis, attackers often use the encryption method's internal structure or brute-force attacks, which consider the encryption algorithm as a black box and systematically test all potential keys. This section first outlines various attack objectives and then discusses different strategies for breaking encryption algorithms.

2.2.1 Attack Models

To evaluate the security of block ciphers, we need to consider both the definition of security and what the attacker can do. The attacker's capabilities, known as the attack model, are typically categorized based on the information the attacker can access.

- **Ciphertext only attack.** The attacker is limited to viewing the ciphertexts only.
- **Known plaintext attack.** The attacker can see both the original messages and their encrypted versions.
- **Chosen plaintext attack.** The attacker can select any message for encryption by the system and subsequently observe the resulting ciphertext. A chosen plaintext attack is more powerful than a known plaintext attack.
- **Chosen ciphertext attack.** The attacker can choose any ciphertext to be decrypted by the system and then see the resulting message.
- **Adaptively chosen plaintext attack.** In this attack model, the attacker can choose the plaintexts based on the previously obtained plaintext-ciphertext pairs.

- **Adaptively chosen ciphertext attack.** In this attack model, the attacker can choose the ciphertexts based on the previously obtained plaintext-ciphertext pairs.

In all the attack models described earlier, the attacker can only get the matching ciphertext (or plaintext) for a known or chosen plaintext (or ciphertext) by making the system generate an output. The system processes requests termed as *queries*, and an entity that responds to these requests is designated as an *oracle*. Additionally, the attacker can have the following model:

- **Single Key Attack.** In this model, the attacker acquires the data related to a fixed secret key.
- **Related Key Attack.** In this model, the attacker can get several ciphertexts (resp. plaintexts) for a single plaintext (resp. ciphertext), and these are encrypted (resp. decrypted) using different keys. It is important to note that the attacker knows how these keys are related.

2.2.2 Goal of the Attacker

Once the adversary has collected an ample amount of data relevant to a specific scenario and within a defined framework, their objective is to pursue one of the following goals:

- **Distinguishing attack:** The distinguishing attack targets the differentiation between data acquired from an actual encryption or decryption oracle and that from an idealized oracle. The idealized oracle generates a fixed-length string chosen uniformly at random, challenging the attacker to discern distinct output characteristics.
- **Key Recovery Attack:** In this scenario, the attacker tries to retrieve the secret key used to encrypt the messages.

Complexity of Cryptanalysis.

Three crucial factors `Data`, `Time`, and `Memory` [40] are employed to assess the effectiveness of an attack.

- **Data Complexity.** The total number of queries made to the encryption (or decryption) oracle to execute an attack in any attack model defines its data complexity.
- **Time Complexity.** The total number of operations required to execute an attack, which includes both online operations (interactions with an oracle or system) and offline computations (such as analysis or search operations). Time complexity captures the overall computational effort needed for the attack to succeed.
- **Memory Complexity.** The total amount of memory required to store all intermediate values, state variables, or data needed for the attack. This includes temporary storage used for calculations and any necessary data structures. For statistical attacks, it typically involves on-the-fly generation of data rather than storing large sets, such as plaintext-ciphertext pairs.

2.2.3 Differential Cryptanalysis

Differential cryptanalysis is widely recognized as a powerful technique for analyzing symmetric key algorithms. It was originally introduced by Biham and Shamir [41] and was initially employed in the analysis of the block cipher DES. Since then, this method has been utilized to scrutinize various other cryptographic algorithms. In essence, differential cryptanalysis focuses on the discrepancies between two distinct computations.

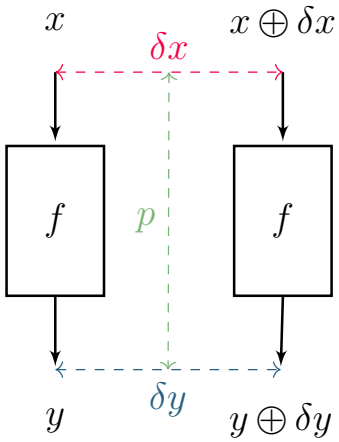


Figure 2.6: General definition of a differential

Basic Concept and Definitions

The difference between two values x and x', denoted as δx, is defined by $\delta x = x \oplus x'$. When these values x and x' undergo processing by the function f, the resulting output difference δy is expressed as:

$$\delta y = f(x) \oplus f(x').$$

This formulation illustrates how δy represents the difference in the outputs of f for inputs x and x'. Differential cryptanalysis involves comparing differences in input and output values of a block cipher E_k. It then assesses the probability that the event $\delta x \xrightarrow{E_k} \delta y$ occurs when x is randomly chosen from a uniform distribution, where δx is the input difference, and δy is the output difference. More formally, we have the following definition:

Definition 2.1 (Probability of Difference Propagation [40]) *Consider a function $\mathcal{F}$ such that n denotes the bit-length of its input. The probability that a specific input difference δx yields a specific output difference δy via the function $\mathcal{F}$ is determined by*

$$\Pr[\delta x \xrightarrow{\mathcal{F}} \delta y] = \frac{\#\{x \in \{0,1\}^n \mid \mathcal{F}(x) \oplus \mathcal{F}(x \oplus \delta x) = \delta y\}}{2^n}.$$

This probability quantifies how often the difference δx in the input of $\mathcal{F}$ causes the difference δy in its output.

Differential Propagation

This section details how differences propagate through the fundamental operations commonly used in block ciphers.

Proposition 2.1 (Differential Propagation through Linear Functions) *Consider L as a linear function, and let δx denote any input difference. Then, the following probabilities hold:*

1. *For any difference δx, we have*

$$\Pr[\delta x \xrightarrow{L} L(\delta x)] = 1$$

   This states that the input difference δx always propagates to $L(\delta x)$ through the linear function L.

2. *For any difference δy where $\delta y \neq L(\delta x)$,*

$$\Pr[\delta x \xrightarrow{L} \delta y] = 0$$

   This indicates that any other output difference δy apart from $L(\delta x)$ cannot result from the input difference δx through the linear function L.

It is crucial to emphasize that for any sequence of linear computations, the resulting output difference can be deterministically computed from the input difference. However, when a function f does not adhere to the linearity property, meaning there exists a pair (a, b) such that $f(a) + f(b) \neq f(a + b)$, no efficient method is currently known to derive high probability differential propagation. Therefore, it is necessary to consider all potential scenarios as defined in the probability of differential propagation (Definition 2.1).

Proposition 2.2 (Differential Propagation through S-box) *Let S be a n-bit S-box, and δx, δy are the input and output differences w.r.t the function S. Then, we have,*

$$D_S(\delta x, \delta y) = |\{x \in \mathbb{F}_2^n \ : \ S(x) \oplus S(x \oplus \delta x) = \delta y\}|$$

where D_S is a $2^n \times 2^n$ table (Difference Distribution Table) to capture the distribution of the differences w.r.t the function S-box.

Differential Distinguisher and Key Recovery Attack

Let E denote an n-bit block cipher, and consider an r-round differential $(\delta x_0, \delta x_r)$ with probability p. Differential cryptanalysis is typically a chosen plaintext or ciphertext attack. The attack can be divided into two main phases: *the distinguishing phase* and the key recovery phase.

Distinguishing Phase

- The attacker chooses a random plaintext $P_0 = x_0$ and constructs $P_0' = x_0' = x_0 \oplus \delta x_0$.
- The encryption oracle is queried with P_0 and P_0', producing the ciphertexts C_0 and C_0'.
- This process is repeated N times to obtain N plaintext-ciphertext pairs (P_i, C_i) and (P_i', C_i'), where $i = 1, 2, \ldots, N$.
- Out of N pairs, approximately Np pairs are expected to satisfy $C \oplus C' = \delta x_r$, where δx_r is the expected ciphertext difference.

Key Recovery Phase. To extend the r-round differential distinguisher into a $(r + k)$-round key recovery attack, the attacker exploits the additional rounds to deduce partial information about the subkey(s) involved. This process can be generalized as follows:

- **Adding Rounds:** The attacker can add key recovery rounds either before or after the r-round differential, depending on the cipher's structure and the differential's propagation properties. The number of key recovery rounds (k) is not limited to 1 and depends on the cipher's design and the attacker's strategy.
- **Plaintext-Ciphertext Collection:** The attacker collects N plaintext-ciphertext pairs (P_i, C_i) and their corresponding difference pairs $(P_i' = P_i \oplus \delta x_0, C_i')$, ensuring that the ciphertext differences are analyzed for possible key candidates.
- **Subkey Guessing:** For each additional round:
  1. The attacker guesses a portion of the subkey K_j, where j denotes the specific round's subkey.
  2. Using the guessed subkey, the attacker partially decrypts (or encrypts) the ciphertext (or plaintext) to reverse the added round and check if the differential holds.
  3. A counter mechanism can be employed to track the number of valid guesses for each key candidate, where the most frequent candidate is identified as the correct subkey.
- **Master Key Recovery:** The subkey guesses are combined to recover the full master key or a significant portion of it. If the cipher uses the same subkey for all rounds (i.e., subkey size equals the master key size), the attacker may require alternative strategies, such as exploiting weak keys or structural properties of the cipher, as storing all possible pairs may not be feasible.

Example: A Simplified Key Recovery Scenario. To illustrate the concept, consider a simple $(r + 1)$-round key recovery attack:

- The attacker collects N plaintext-ciphertext pairs (P_i, C_i) and $(P_i' = P_i \oplus \delta x_0, C_i')$.

- Assuming K_{r+1} is the subkey for the last round, the attacker guesses K_{r+1} and partially decrypts C_i and C'_i through the last round. Specifically, for every guessed value of K_{r+1} and for each pair (P_i, C_i) in the collected plaintext-ciphertext pairs (P_i, C_i) and $(P_i \oplus \delta x_0, C'_i)$, the attacker performs decryption using K_{r+1}, computing $x_r = f^{-1}(C_i \oplus K_{r+1})$ and $x'_r = f^{-1}(C'_i \oplus K_{r+1})$.

- The attacker initializes an array named `counter` where each element `counter`$[i]$ is set to 0 for $i = 0, 1, \ldots, 2^\kappa - 1$, with κ representing the bit-length of the $r+1$-th round key K_{r+1}. Now, if $x_r \oplus x'_r = \delta x_r$, the attacker increments the corresponding counter `counter`$[K_{r+1}]$ by 1. Eventually, one of the counter values will significantly surpass the others (specifically reaching c), indicating the correct candidate for K_{r+1}. It is important to note that, using a counter is one of the possibilities.

Note: The example provided here is deliberately simplified to explain the core concept. In practice, key recovery attacks can involve multiple rounds, more complex differential paths, and additional considerations such as memory constraints and optimized search techniques.

2.2.4 Impossible Differential Cryptanalysis

Impossible differential cryptanalysis, a variant of differential cryptanalysis, utilizes a differential propagation that can never occur. The basic definition of impossible differential characteristics is as follows:

Definition 2.2 (Impossible Differential Characteristics [40]) *Let δx and δy denote the input and output difference of function $\mathcal{F}$. A pair $(\delta x, \delta y)$ is termed an impossible differential for $\mathcal{F}$ if*

$$\Pr[\delta x \xrightarrow{\mathcal{F}} \delta y] = 0$$

Example. A trivial example of an impossible differential characteristic is that a non-zero input difference cannot propagate to a zero output difference through a bijective function $\mathcal{F}$.

Brief Overview. Lars Knudsen [42] first employed the impossible differential attack against the 128-bit block cipher DEAL. Furthermore, in [43], the authors used the term impossible differential and demonstrated a technique for constructing an impossible differential over $2r$ rounds using two r-round differentials with a probability of 1. In this method, the initial differential converts the difference δx to δz in one direction, while the subsequent differential converts δy to $\delta z'$ in the opposite direction. Should the differences fail to align, the $2r$-round differential $\delta x \rightarrow \delta y$ is identified as an impossible differential. This technique is known as the *miss-in-the-middle* approach. Figure 2.7 illustrates an example of *miss-in-the-middle* attack on AES. Although we illustrate this example using AES, a similar structure for the meet-in-the-middle attack can be observed in other AES-like block ciphers. The variations may arise primarily from differences in the branch number of the MixColumns operation and the specifics of the ShiftRows operation. An impossible differential attack leverages impossible differential characteristics within a cipher to deduce the correct key, eliminating incorrect keys that lead to this improbable differential. In detail, the attack

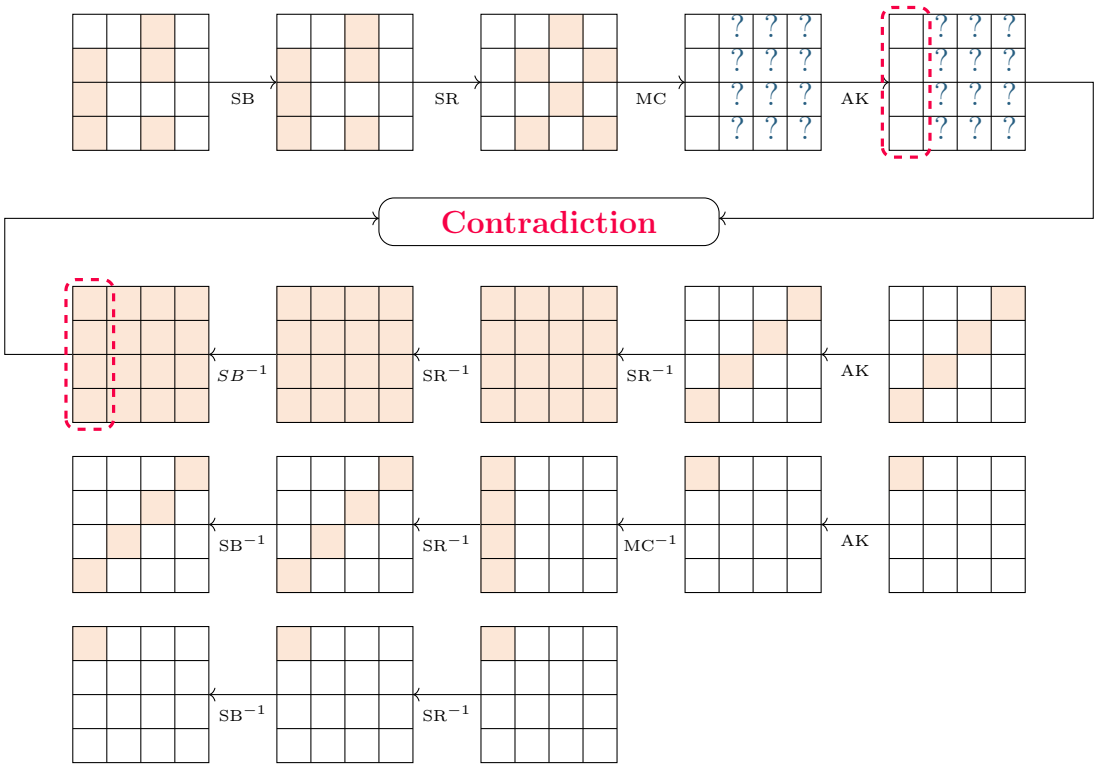


Figure 2.7: Overview of miss-in-the-middle concept on AES

methodology starts by finding an r-round impossible differential. We enhance the impossible differential attack by incorporating additional rounds before and after. If a potential key produces the impossible differential during partial encryption or decryption of a given pair, that key is discarded to narrow down potential matches. The objective is to eliminate as many incorrect keys as feasible before conducting a systematic search to find the correct key.

Key Recovery and Complexity Analysis in Impossible Differential Attacks

Let us consider a block cipher E with a block size of n bits and a key size of κ bits. We consider pairs (X, X') with $X, X' \in \mathbb{F}_2^n$ and denote their difference by $\Delta = X \oplus X'$. Then, $\Pr(\Delta_{\text{U}} \to \Delta_{\text{L}})$ denotes the expected differential probability that an input pair (X, X') with $X \oplus X' = \Delta_{\text{U}}$ yields an output pair with $E(X) \oplus E(X') = \Delta_{\text{L}}$, where E is implicit from the context. More generally, we use the same notation to denote bitwise truncated differences $\Delta \subseteq \mathbb{F}_2^n$, and the corresponding probabilities averaged over all input differences in the set. Given a (truncated) input difference Δ_{U}, we refer to the least truncated output difference Δ_{L} such that $\Pr(\Delta_{\text{U}} \to \Delta_{\text{L}}) = 1$ as propagation with probability one or deterministic propagation. If $\Pr(\Delta_{\text{U}} \to \Delta_{\text{L}}) = 0$, we write $\Delta_{\text{U}} \nrightarrow \Delta_{\text{L}}$ and call this an impossible differential. Suppose there exists an impossible differential $\Delta_{\text{U}} \nrightarrow \Delta_{\text{L}}$ for r_{D} rounds of E (denoted as E_{D}). To perform key recovery, as illustrated in Figure 2.8, we extend the distinguisher by a few rounds at both ends. Let E_{B} and E_{F} represent the rounds added before

and after E_{D}, respectively, with r_{B} and r_{F} denoting their respective numbers, such that $E = E_{\mathrm{F}} \circ E_{\mathrm{D}} \circ E_{\mathrm{B}}$. Subsequently, we propagate the difference Δ_{U} (and Δ_{L}) through E_{B}^{-1} (and E_{F}) with probability one to obtain the truncated difference Δ_{B} (and Δ_{F}). Here, $|\Delta_{\mathrm{B}}|$ and $|\Delta_{\mathrm{F}}|$ denote the number of non-fixed bit differences in Δ_{B} and Δ_{F}, respectively. Assume that $\Pr(\Delta_{\mathrm{B}} \rightarrow \Delta_{\mathrm{U}}) = 2^{-c_{\mathrm{B}}}$ and $\Pr(\Delta_{\mathrm{L}} \leftarrow \Delta_{\mathrm{F}}) = 2^{-c_{\mathrm{F}}}$. In the context of (impossible) differential key recovery, c_{B} and c_{F} are typically referred to as the number of bit filters that should be satisfied for differential transitions $\Delta_{\mathrm{B}} \rightarrow \Delta_{\mathrm{U}}$ and $\Delta_{\mathrm{L}} \leftarrow \Delta_{\mathrm{F}}$, respectively. As illustrated in Figure 2.8, assume that the key bits $k_{\mathrm{B}} \cup k_{\mathrm{F}}$ are involved in deriving the difference Δ_{U} and Δ_{L} from Δ_{B} and Δ_{F}, respectively. With these parameters established, we divide the key recovery of an ID attack into three steps:

- **Pair Generation.** In this step, we generate N plaintext pairs (P, P') such that $P \oplus P' \in \Delta_{\mathrm{B}}$ and $E(P) \oplus E(P') \in \Delta_{\mathrm{F}}$. The problem of finding such pairs is known as the *limited birthday problem*. The complexity of this step is (see [44]) $T_0 := \max\left\{ \min_{\Delta \in \{\Delta_{\mathrm{B}}, \Delta_{\mathrm{F}}\}} \left\{\sqrt{N 2^{n+1-|\Delta|}}\right\}, N 2^{n+1-|\Delta_{\mathrm{B}}|-|\Delta_{\mathrm{F}}|} \right\}$.
- **Guess-and-Filter.** In this step, we eliminate the incorrect candidates for $k_{\mathrm{B}} \cup k_{\mathrm{F}}$ by checking whether a candidate for involved key bits yields the impossible differential for at least one of the N pairs. We typically use the *early abort* technique to perform this step [45]: we split $k_{\mathrm{B}} \cup k_{\mathrm{F}}$ into several subsets and guess them one by one. At each step, we check some new bit filters and discard a portion of the pairs that do not satisfy the bit filters. The correct key guess never suggests an impossible differential for any pairs. Thus, we perform N partial encryptions/decryptions for the correct key guess. However, a wrong key guess may suggest an impossible differential for some pairs. The more pairs we have, the more likely a wrong key guess suggests an impossible differential for at least one of the pairs. A lower bound for the complexity of this step is (see [44]) $T_1 + T_2 = N + 2^{|k_{\mathrm{B}} \cup k_{\mathrm{F}}|} \frac{N}{2^{c_{\mathrm{B}}+c_{\mathrm{F}}}}$ partial encryptions.
- **Exhaustive Search.** The probability that an incorrect key succeeds in the guess-and-filter procedure is $P = \left(1 - 2^{-(c_{\mathrm{B}}+c_{\mathrm{F}})}\right)^N$, which means the expected number of wrong keys that pass the guess-and-filter step is $P \cdot 2^{|k_{\mathrm{B}} \cup k_{\mathrm{F}}|}$. Considering that $\kappa - |k_{\mathrm{B}} \cup k_{\mathrm{F}}|$ key bits are not involved in the guess-and-filter step, we should brute-force a key space of size $T_3 = 2^{\kappa - |k_{\mathrm{B}} \cup k_{\mathrm{F}}|} \cdot P \cdot 2^{|k_{\mathrm{B}} \cup k_{\mathrm{F}}|} = P \cdot 2^{\kappa}$ to uniquely retrieve the correct key.

Suppose we assume that C_E represents the cost of executing E, and $C_{E'}$ represents the proportion of the cost for executing E_{B} and E_{F} relative to the entire encryption process. In that case, the overall time complexity of the impossible differential key recovery is: $T_{\mathrm{tot}} = (T_0 + (T_1 + T_2) C_{E'} + T_3) C_E$. If we consider the number of encryption queries as the data complexity, then T_0 represents the data complexity. To ensure that the data complexity remains below the size of the entire code book, one should have $T_0 < 2^n$, and to keep the time complexity less than brute force, one requires $T_{\mathrm{tot}} < 2^{\kappa}$.

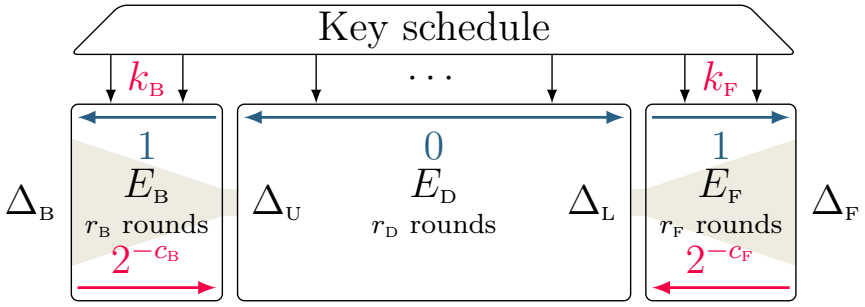


Figure 2.8: Overview and parameters of impossible differential attacks.

2.2.5 Linear Cryptanalysis

Matsui [46] introduced linear cryptanalysis as a novel known-plaintext attack on DES. This method, presented at Eurocrypt 1993, utilizes probabilistic linear relations or linear approximations. The foundational idea of linear approximations was initially applied in [47].

Let $\mathbb{F}_2^m$ be the finite field with 2^m elements, and '+' denote the addition in this field. Let $\mathcal{F} : \mathbb{F}_2^m \to \mathbb{F}_2^m$ be an iterative permutation. Now, in linear cryptanalysis, the attacker attempts to find masks $u, v \in \mathbb{F}_2^m$ such that $v^{\mathbf{T}}\mathcal{F}(x) = u^{\mathbf{T}}x$ holds significantly more or less often than is expected for an ideal cipher where $u^{\mathbf{T}}x = \sum_{i=1}^{m} u_i x_i$. Hence, (u, v) is called a linear approximation of $\mathcal{F}$. Now, the bias ε of a linear approximation is as follows:

$$\varepsilon = \Pr_x[v^{\mathbf{T}}\mathcal{F}(x) = u^{\mathbf{T}}x] - \frac{1}{2}$$

where x is uniformly distributed on $\mathbb{F}_2^m$. In this context, we can define the correlation of a linear approximation of $\mathcal{F}$ as $c = 2\varepsilon$. Therefore, the correlation c satisfies

$$c = 2\Pr_x[v^{\mathbf{T}}\mathcal{F}(x) = u^{\mathbf{T}}x] - 1$$

In linear cryptanalysis, the goal for the attacker is to identify a linear approximation where $|\varepsilon|$ is maximized. Discovering an appropriate linear approximation for a block cipher $\mathcal{F}$ directly provides a known-plaintext distinguisher. Calculating $|\varepsilon|$ is generally challenging because the set of plaintexts $\mathbb{F}_2^m$ is extremely large for $m \geqslant 64$. However, if $\mathcal{F} = \mathcal{F}_r \circ \ldots \circ \mathcal{F}_1$, where each function $\mathcal{F}_i$ is relatively straightforward to analyze, linear approximations can be evaluated round-by-round using linear trails. A linear trail refers to a sequence of linear approximations that describe how biases in input-output correlations propagate through the individual rounds of a cipher. The biases, or deviations from uniform probability, are tracked for every round, providing a way to evaluate the overall effectiveness of a linear approximation across multiple rounds. Linear trails are the backbone of linear cryptanalysis, much like differential trails or characteristics are for differential cryptanalysis.

In the context of differential cryptanalysis, a differential trail tracks how specific input differences propagate through the cipher to produce output differences, with each step corresponding to a specific probability. These trails are used to assess the cipher's vulnerability to attacks based on these probabilities. Similarly, in linear cryptanalysis, a linear trail tracks how specific linear approximations propagate through the rounds, with the bias (deviation from uniform probability) associated with each round being a key parameter.

Furthermore, Matsui proposed the following lemma to calculate the correlation of the combined approximations.

Lemma 2.1 (Piling-up Lemma [46]) *For all independent random variables $z_1, z_2, \ldots, z_r$ on $\mathbb{F}_2$ with $c_i = 2\Pr[z_i = 0] - 1$, it holds that*

$$2\Pr\left[\sum_{i=1}^{r} z_i = 0\right] - 1 = \prod_{i=1}^{r} c_i$$

.

Let $x_{i+1} = \mathcal{F}(x_i)$, for $i = 1, 2, \ldots r$ with x_1 uniform random on $\mathbb{F}_2^m$, and $(v_1, \ldots, v_r, v_{r+1})$ be a linear trail. Therefore, the attacker can apply Lemma 2.1 with $z_i = v_{i+1}^{\mathbf{T}} \mathcal{F}_i(x_i) + v_i^{\mathbf{T}} x_i$ to estimate the correlation of the linear approximation (v_1, v_{r+1}). This lemma assumes that the random variables z_i are independent, and we have

$$\sum_{i=1}^{r} z_i = v_{r+1}^{\mathbf{T}} \mathcal{F}_r(x_r) + v_1^{\mathbf{T}} x_1$$

Furthermore, in [48], the authors presented an alternative description of linear cryptanalysis using the concept of correlation matrices. First, the authors in [48] define correlation coefficients as follows:

Definition 2.3 (Correlation coefficient [48]) *The correlation between two Boolean functions $f, g : \mathbb{F}_2^m \to \mathbb{F}_2$ is the quantity*

$$C(f, g) = 2\Pr_x[f(x) = g(x)] - 1$$

for x uniformly distributed on $\mathbb{F}_2^m$.

Given a function $\mathcal{F} : \mathbb{F}_2^m \to \mathbb{F}_2^m$, the authors in [48] define a real matrix $C^{\mathcal{F}}$ with entries

$$C^{\mathcal{F}}_{v,u} = C(\ell_v \circ \mathcal{F}, \ell_u)$$

where $\ell_u(x) = u^{\mathbf{T}} x$, with $u \in \mathbb{F}_2^m$. $C^{\mathcal{F}}$ is called the correlation matrix of $\mathcal{F}$. Now, if $\mathcal{F} : \mathbb{F}_2^m \to \mathbb{F}_2^m$ is an iterative permutation such that $\mathcal{F} = \mathcal{F}_r \circ \ldots \circ \mathcal{F}_1$, where $\mathcal{F}_i : \mathbb{F}_2^m \to \mathbb{F}_2^m$ for $i = 1, 2, \ldots r$, then

$$C^{\mathcal{F}} = C^{\mathcal{F}_r} \times \cdots \times C^{\mathcal{F}_1}$$

A linear trail $V = (v_1, \ldots, v_{r+1})$ is the concatenation of r 1-round linear approximations, where (v_i, v_{i+1}) is the approximation of $\mathcal{F}_i$. The correlation of a linear trail V is defined as

$$C_V = \prod_{i=1}^{r} C(\ell_{v_{i+1}} \circ \mathcal{F}_i, \ell_{v_i})$$

A linear approximation (α, β) of a block cipher is called a linear hull which is defined as follows:

$$C(\ell_\beta \circ \mathcal{F}, \ell_\alpha) = \sum_{V : v_1 = \alpha, v_{r+1} = \beta} C_V.$$

Correlation over basic operations

We begin by examining the correlation of approximations over four fundamental operations frequently found in block ciphers: the XOR operation, the branching operation, the linear mapping, and the S-box.

Proposition 2.3 (Approximation of `Branching`) *Let $f : \mathbb{F}_2 \to \mathbb{F}_2 \times \mathbb{F}_2$ such that $f(x) = (x, x)$. Then,*

$$C(\ell_v \circ f, \ell_u) \neq 0 \text{ if and only if } u = v_1 + v_2 \bmod 2$$

where u and $v = (v_1, v_2)$ are binary variables corresponding to input and output linear masks, respectively.

Proposition 2.4 (Approximation of `XOR`) *Let $f : \mathbb{F}_2 \times \mathbb{F}_2 \to \mathbb{F}_2$ such that $f(x_1, x_2) = x_1 \oplus x_2$. Then,*

$$C(\ell_v \circ f, \ell_{(u_1,u_2)}) \neq 0 \text{ if and only if } u_1 = u_2 = v$$

where (u_1, u_2), and v are binary variables corresponding to input and output linear masks, respectively.

Proposition 2.5 (Approximation of a linear map) *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a linear map such that $f(x) = Mx$. Then we have,*

$$C(\ell_v \circ f, \ell_u) = \begin{cases} 1, & \text{if } u = M^{\mathbf{T}} v \\ 0, & \text{if } u \neq M^{\mathbf{T}} v \end{cases}$$

where $u = (u_1, u_2, \ldots, u_n) \in \mathbb{F}_2^n$, and $v = (v_1, v_2, \ldots, v_n) \in \mathbb{F}_2^n$ are corresponding to input and output linear masks, respectively.

Proposition 2.6 (Approximation of S-box) *Let S be a n-bit S-box and u, v is the input and output linear masks w.r.t the function S. Then we have,*

$$L_S(u, v) = |\{x \in \mathbb{F}_2^n \ : \ u^{\mathbf{T}} x = v^{\mathbf{T}} S(x)\}| - 2^{n-1}$$

where L_S is a $2^n \times 2^n$ table (Linear Approximation Table) to capture linear approximation w.r.t the function S-box.

Zero-Correlation Linear Distinguisher

The zero-correlation linear hull concept was first introduced in [49]. As explained in [49], the attacker designs zero-correlation linear hulls where $C_V = 0$ for every linear trail V within the hull and all-round functions that depend on the secret key in a block cipher. Multiple zero-correlation cryptanalysis was introduced in [50], where a new distinguisher was proposed based on numerous zero-correlation approximations in vulnerable ciphers. Later, [51] presented an even more powerful tool, the multidimensional zero-correlation distinguisher.

Since this thesis does not utilize these methods, they are not explained in detail here. For comprehensive information on these topics, please refer to [50], [51].

Zero-correlation linear distinguishing attacks aim to identify linear approximations with zero correlation, enabling the distinction between a block cipher and a random permutation. These attacks are the counterpart to impossible differential attacks in linear analysis. Hence, one can build the zero-correlation linear distinguisher based on the *miss-in-the-middle* approach. In [52], the authors used the matrix method for finding zero-correlation linear approximation based on *miss-in-the-middle* attack. They examined the linear patterns of input and output masks in the intermediate rounds to determine if any linear characteristics with non-zero correlation were present. To recover the key using a zero correlation linear approximation distinguisher for part of the cipher, one can employ a technique similar to Matsui's Algorithm 2 [46]. This approach involves partially encrypting or decrypting from the plaintext or ciphertext up to the limits of the identified property. This key recovery approach is mainly used in most zero-correlation attacks. This thesis limits our focus to identifying zero-correlation linear distinguishers for AndRX ciphers. Therefore, we do not provide a detailed explanation of the key recovery process in zero-correlation attacks.

2.2.6 Integral Distinguisher

In [53], Daemen *et al.* introduced integral cryptanalysis for evaluating the security of the SQUARE block cipher. The purpose of integral cryptanalysis is to leverage various properties that are highly likely to occur in the common computations used in block ciphers. More precisely, integral cryptanalysis involves tracking a subset of input bits and a corresponding subset of output bits to ensure that the sum of ciphertexts exhibits a balanced property in some of these output bits. This means that the sum is zero for certain output bits, given that a group of input messages takes every possible value for the selected input bits while the remaining input bits are kept constant but arbitrary. Here are the definitions of four fundamental integral properties.

- `ALL` ($\mathcal{A}$): The multi set shows each value the same number.
- `BALANCE` ($\mathcal{B}$): The XOR of all texts in the multi set is 0.
- `CONSTANT` ($\mathcal{C}$): For all texts in the multi set, the value is fixed by a constant.
- `UNKNOWN` ($\mathcal{U}$): The multi set cannot be differentiated from an n-bit random value.

To better understand how these properties propagate through the operations in a block cipher, let us consider a simplified example of an integral distinguisher for a 3-round AES (Figure 2.9). AES operates as a byte-oriented cipher, meaning that its round function processes data at the byte level. For integral analysis, we consider a collection of 256 plaintexts where one byte takes all 256 possible values, while all other bytes remain fixed to the same value across these plaintexts. Let this set of plaintexts be denoted as $\mathcal{P}$. In this case, the first byte of the set $\mathcal{P}$ exhibits the `ALL` property, whereas the remaining bytes display the `CONSTANT` property (see Figure 2.9). AES consists of several operations in each round, such as SubBytes (S-box), ShiftRows, MixColumns, and AddRoundKey. These operations impact the properties of the input bytes and their propagation as follows:

AddRoundKey (ARK) operation: The ARK operation `XOR`s the state with a round key. By `XOR`ing an unknown or known constant for each of the texts in the set,

- the byte with `ALL` property still satisfies the `ALL` property
- the byte with `CONSTANT` property still satisfies the `CONSTANT` property

SubByte (SB) operation: The SB operation applies a non-linear S-box to each byte of the state. By applying the S-box for each of the texts in the set,

- the byte with `ALL` property still satisfies the `ALL` property
- the byte with `CONSTANT` property still satisfies the `CONSTANT` property

ShiftRows (SR) operation: The SR operation cyclically shifts the rows of the state, which affects the positions of the bytes. Hence, this operation never affects the property inside a byte.

MixColumns (MC) operation: The MixColumns operation mixes the columns of the state by multiplying them with a fixed matrix.

- If there is exactly one byte with the all property in each column, then four output bytes from each column will have the all property.
- If the number of bytes with the all property is exactly four in each column, then four output bytes from each column will have the all property

2.2.7 Division Property

At Eurocrypt 2015, Todo introduced the Division property [54], a groundbreaking technique for discovering integral characteristics that aid in identifying integral distinguishers within block cipher architectures, including Feistel and SPN structures. The author investigated how the division property moves through different operations in block ciphers and formulated generalized algorithms to detect integral distinguishers. This method focuses exclusively on the algebraic degree of the block cipher's nonlinear components. Moreover, at FSE 2016, Todo and Morii [21] presented two varieties of bit-based division properties, namely *conventional bit-based division property* (CBDP) and *bit-based division property using three subsets* (BDPT). The definitions for these properties are as follows.

Definition 2.4 ***(CBDP [21]).*** *Consider a multi set $\mathbb{X} \subseteq \mathbb{F}_2^n$, and a set $\mathbb{K}$ containing n-dimensional bit vectors. The multi set $\mathbb{X}$ possesses the division property $D_{\mathbb{K}}^{1^n}$, if it adheres to the following conditions:*

$$\bigoplus_{x \in \mathbb{X}} x^u = \begin{cases} \textit{unknown}, & \textit{if there exists } k \in \mathbb{K} \textit{ such that } u \succeq k, \\ 0, & \textit{otherwise.} \end{cases}$$

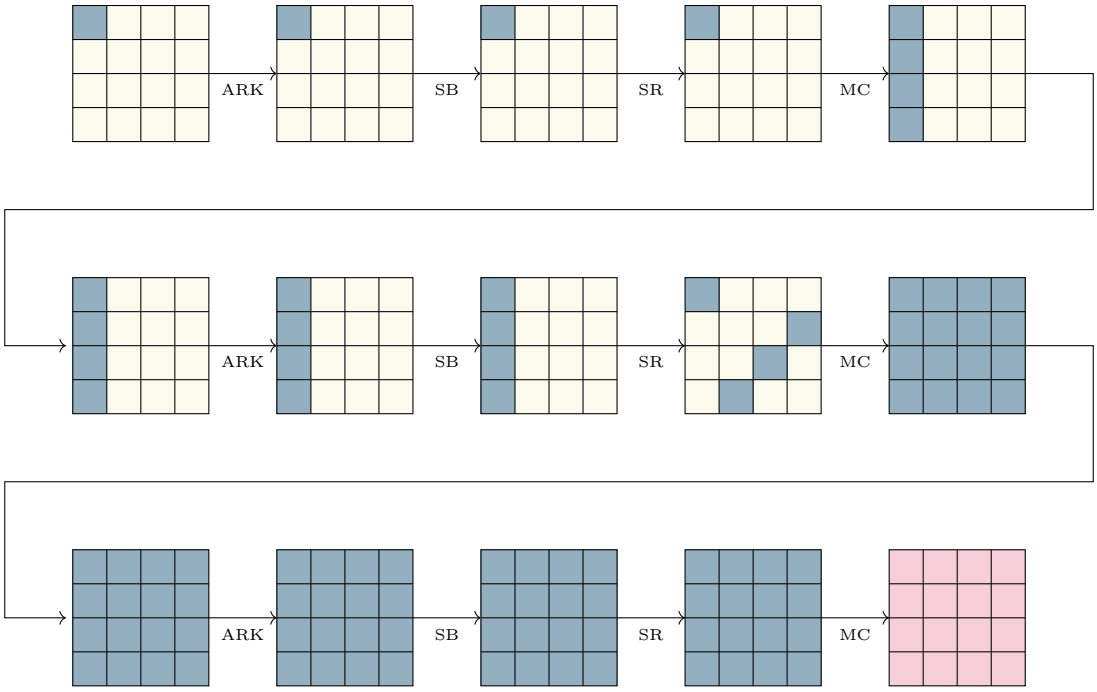


Figure 2.9: Integral distinguisher for 3-round AES. ■ : ALL property. □ : CONSTANT property. ■ : BALANCE property

Definition 2.5 *(**BDPT** [21]) Consider a multi set $\mathbb{X} \subseteq F_2^n$. Suppose $\mathbb{K}$ and $\mathbb{L}$ are two subsets of F_2^n. The multi set $\mathbb{X}$ is said to have the division property $D^{1^n}_{\mathbb{K},\mathbb{L}}$ if it meets the following criteria:*

$$\bigoplus_{x\in\mathbb{X}} x^u = \begin{cases} \textit{unknown}, & \textit{if there exists } k \in \mathbb{K} \textit{ such that } u \succeq k, \\ 1, & \textit{if there exists } \ell \in \mathbb{L} \textit{ such that } u = \ell, \\ 0, & \textit{otherwise.} \end{cases}$$

If there exist $k \in \mathbb{K}$ and $k' \in \mathbb{K}$ such that $k \succeq k'$ in the CBDP $D^{1^n}_{\mathbb{K}}$, the vector k is considered redundant and can be eliminated from $\mathbb{K}$. This process is referred to as ***Reduce*0**$(\mathbb{K})$. Additionally, if there exist $\ell \in \mathbb{L}$ and $k \in \mathbb{K}$ where $\ell \succeq k$, then ℓ is also redundant. This elimination process is denoted as ***Reduce*1**$(\mathbb{K}, \mathbb{L})$. These redundant vectors in $\mathbb{K}$ and $\mathbb{L}$ do not influence the parity of x^u for any u.

In CBDP, the handling of $\mathbb{K}$ follows similar principles in BDPT. This chapter introduces explicitly and explains these rules as they pertain to BDPT. For additional insights, the readers are requested to refer to [21], [23].

BDPT Rule 2.7 (COPY [21]) *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^{n+1}$ such that $F(x_0, \ldots, x_{n-1}) = (x_0, x_0, \ldots, x_{n-1})$. Define multi sets $\mathbb{X}$ and $\mathbb{Y}$ as follows: $\mathbb{X}$ satisfies $D^{1^n}_{\mathbb{K},\mathbb{L}}$, and $\mathbb{Y}$ satisfies*

$D^{1^{n+1}}_{\mathbb{K}',\mathbb{L}'}$. *The sets* $\mathbb{K}'$ *and* $\mathbb{L}'$ *are defined by:*

$$\mathbb{K}' \leftarrow \begin{cases} (0,0,k_1,k_2,\ldots,k_{n-1}), & \text{if } k_0 = 0 \\ (1,0,k_1,k_2,\ldots,k_{n-1}),(0,1,k_1,k_2,\ldots,k_{n-1}), & \text{if } k_0 = 1 \end{cases}$$

$$\mathbb{L}' \leftarrow \begin{cases} (0,0,\ell_1,\ell_2,\ldots,\ell_{n-1}), & \text{if } \ell_0 = 0 \\ (1,0,\ell_1,\ell_2,\ldots,\ell_{n-1}),(0,1,\ell_1,\ell_2,\ldots,\ell_{n-1}),(1,1,\ell_1,\ell_2,\ldots,\ell_{n-1}), & \text{if } \ell_0 = 1 \end{cases}$$

where $k = (k_0,k_1,\ldots,k_{n-1}) \in \mathbb{K}$ *and* $\ell = (\ell_0,\ell_1,\ldots,\ell_{n-1}) \in \mathbb{L}$.

BDPT Rule 2.8 (XOR [21]) *Let* $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^{n-1}$ *be defined as* $F(x_0,x_1,\ldots,x_{n-1}) = (x_0 \oplus x_1, x_2, \ldots, x_{n-1})$. *Define multi sets* $\mathbb{X}$ *and* $\mathbb{Y}$ *as follows:* $\mathbb{X}$ *satisfies* $D^{1^n}_{\mathbb{K},\mathbb{L}}$, *and* $\mathbb{Y}$ *satisfies* $D^{1^{n-1}}_{\mathbb{K}',\mathbb{L}'}$, *where:*

- $\mathbb{K}'$ *is computed from* $\mathbb{K}$ *such that* $(k_0,k_1) \in \{(0,0),(1,0),(0,1)\}$*:*

$$\mathbb{K}' \leftarrow (k_0 + k_1, k_2, k_3, \ldots, k_{n-1}).$$

- $\mathbb{L}'$ *is computed from* $\mathbb{L}$ *such that* $(\ell_0,\ell_1) \in \{(0,0),(1,0),(0,1)\}$*:*

$$\mathbb{L}' \overset{x}{\leftarrow} (\ell_0 + \ell_1, \ell_2, \ell_3, \ldots, \ell_{n-1}),$$

  where $\mathbb{L} \overset{x}{\leftarrow} \ell$ *denotes:*

$$\mathbb{L} = \begin{cases} \mathbb{L} \cup \{\ell\}, & \text{if } \ell \notin \mathbb{L}, \\ \mathbb{L} \backslash \{\ell\}, & \text{if } \ell \in \mathbb{L}. \end{cases}$$

BDPT Rule 2.9 (AND [21]) *Let* $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^{n-1}$ *be defined as* $F(x_0,x_1,\ldots,x_{n-1}) = (x_0 \wedge x_1, x_2, \ldots, x_{n-1})$. *Define multi sets* $\mathbb{X}$ *and* $\mathbb{Y}$ *as follows:* $\mathbb{X}$ *satisfies* $D^{1^n}_{\mathbb{K},\mathbb{L}}$, *and* $\mathbb{Y}$ *satisfies* $D^{1^{n-1}}_{\mathbb{K}',\mathbb{L}'}$, *where:*

- $\mathbb{K}'$ *is derived from* $\mathbb{K}$ *such that:*

$$\mathbb{K}' \leftarrow \left(\left\lceil \frac{k_0 + k_1}{2} \right\rceil, k_2, k_3, \ldots, k_{n-1} \right),$$

  for $k = (k_0,k_1,\ldots,k_{n-1}) \in \mathbb{K}$.

- $\mathbb{L}'$ *is derived from* $\mathbb{L}$ *such that:*

$$\mathbb{L}' \leftarrow \left(\left\lceil \frac{\ell_0 + \ell_1}{2} \right\rceil, \ell_2, \ell_3, \ldots, \ell_{n-1} \right),$$

  for $\ell = (\ell_0,\ell_1,\ldots,\ell_{n-1}) \in \mathbb{L}$, *where* $(\ell_0,\ell_1) \in \{(0,0),(1,1)\}$.

BDPT Rule 2.10 (XOR Operation with Key [21]) *Let* $F_k : \mathbb{F}_2^n \to \mathbb{F}_2^n$ *such that* $F_k(x_0, \ldots, x_{n-1}) = (x_0, \ldots, x_i \oplus k, \ldots, x_{n-1})$*, where* k *denotes the secret key. Define multi sets* $\mathbb{X}$ *and* $\mathbb{Y}$ *as follows:* $\mathbb{X}$ *satisfies* $D^{1^n}_{\mathbb{K},\mathbb{L}}$*, and* $\mathbb{Y}$ *satisfies* $D^{1^n}_{\mathbb{K}',\mathbb{L}'}$*. The sets* $\mathbb{K}'$ *and* $\mathbb{L}'$ *are defined by:*

$$\begin{cases} \mathbb{L}' \leftarrow \ell \quad \text{for } \ell \in \mathbb{L}, \\ \mathbb{K}' \leftarrow k \quad \text{for } k \in \mathbb{K}, \\ \mathbb{K}' \leftarrow (\ell_0, \ell_1, ..., \ell_i \vee 1, ..., \ell_{n-1}) \quad \text{for } \ell \in \mathbb{L} \text{ with } \ell_i = 0. \end{cases}$$

BDPT Rule 2.11 (S-box Characteristics [23]) *Consider an S-box that maps* $\mathbb{F}_2^n$ *to* $\mathbb{F}_2^n$*, where* $x = (x_0, \ldots, x_{n-1})$ *and* $y = (y_0, \ldots, y_{n-1})$ *denote the input and output variables, respectively. Each output bit* y_i*, for* $i \in \{0, 1, \ldots, n-1\}$*, is expressed as a Boolean function of the input bits* $(x_0, x_1, \ldots, x_{n-1})$*. Assuming the input BDPT of the S-box is* $D^{1^n}_{\mathbb{K},\mathbb{L}=\{\ell\}}$*, the corresponding output BDPT is* $D^{1^n}_{\boldsymbol{Reduce0}(\underline{\mathbb{K}}),\boldsymbol{Reduce1}(\underline{\mathbb{K}},\underline{\mathbb{L}})}$*, where*

$$\begin{cases} \underline{\mathbb{K}} = \{u' \in \mathbb{F}_2^n \ : \ \text{for any } u \in \mathbb{K}, \text{if } y^{u'} \text{ contains any term } x^v \text{ with } v \succeq u\}, \\ \underline{\mathbb{L}} = \{u \in \mathbb{F}_2^n \ : \ y^u \text{ contains the term } x^\ell\}. \end{cases}$$

Let $D^{1^n}_{\mathbb{K},\mathbb{L}=\{\ell^0,\ldots,\ell^{r-1}\}}$ *and* $D^{1^n}_{\mathbb{K}',\mathbb{L}'}$ *denote the input and output BDPT of the S-box, respectively, where* $\ell^i = (\ell^i_0, \ldots, \ell^i_{n-1})$ *for all* $i = 0, \ldots, (r-1)$*. The output BDPT* $D^{1^n}_{\mathbb{K}',\mathbb{L}'_i}$ *is derived from the corresponding input BDPT* $D^{1^n}_{\mathbb{K},\mathbb{L}=\{\ell^i\}}$ *for* $i = 0, 1, \ldots, r-1$*. Thus,*

$$\mathbb{L}' = \{\ell \ : \ \ell \text{ appears an odd number of times in sets } \mathbb{L}'_0, \ldots, \mathbb{L}'_{r-1}\}.$$

2.2.8 Cube Attacks

The cube attack was proposed by Dinur and Shamir [31] in 2009. Let $f(x, v)$ be a Boolean function where one can write secret variables $x = (x_0, x_1, \ldots, x_{n-1})$ and public variables $v = (v_0, v_1, \ldots, v_{m-1})$. The Algebraic Normal Form (ANF) of the Boolean function can be expressed as follows:

$$f(x,\, v) = \bigoplus_{u \in \mathbb{F}_2^{n+m}} a^f_u (x \parallel v)^u.$$

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_{|\mathcal{I}|}\}$ denote a subset of indices, which one can call cube indices. Define the monomial $t_\mathcal{I}$ for these indices as $t_\mathcal{I} = v_{i_1} \cdot v_{i_2} \cdots v_{i_{|\mathcal{I}|}}$. Now, $f(x, v)$ can be written as:

$$f(x, v) = t_\mathcal{I} \cdot p(x, v) + q(x, v).$$

It is important to note that, $p(x, v)$ is a polynomial that does not contain any common variable with $t_\mathcal{I}$, and each term in $q(x, v)$ excludes at least one variable from $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|\mathcal{I}|}}\}$. Let $C_\mathcal{I}$ denote the cube determined by indices $\mathcal{I}$. This cube encompasses $2^{|\mathcal{I}|}$ combinations of values for variables $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|\mathcal{I}|}}\}$, while the remaining variables are fixed. The sum of $f(x, v)$ over all configurations in cube $C_\mathcal{I}$ is expressed as:

$$\bigoplus_{C_\mathcal{I}} f(x, v) = \bigoplus_{C_\mathcal{I}} (t_\mathcal{I} \cdot p(x, v)) + \bigoplus_{C_\mathcal{I}} q(x, v) = p(x, v).$$

This outcome arises because $t_{\mathcal{I}} = 1$ for exactly one configuration within $C_{\mathcal{I}}$, while each term in $q(x, v)$ excludes at least one variable from $\{v_{i_1}, v_{i_2}, \ldots, v_{i_{|\mathcal{I}|}}\}$. The cube attack primarily focuses on determining the function $p(x, v)$, referred to as the superpoly of the cube $C_{\mathcal{I}}$.

Cube attacks and division property.

The division property evolved as a more detailed aspect following the integral property, initially employed to scrutinize the integral distinguisher. The discussion that follows explores the connection between the public functions' ANF and division property:

Definition 2.6 (Three-Subset Division Property without Unknown Subset [33]) *Consider the multi sets $\mathbb{X}$, and $\tilde{\mathbb{L}}$ whose elements are in $\mathbb{F}_2^n$. The multi set $\mathbb{X}$ is said to have the three-subset division property without an unknown subset ($\mathcal{T}_{\tilde{\mathbb{L}}}^{1^n}$) if it meets the following criterion:*

$$\bigoplus_{x \in \mathbb{X}} x^u = \begin{cases} 1, & \text{if } u \text{ appears an odd number of times in } \tilde{\mathbb{L}} \\ 0, & \text{if } u \text{ appears an even number of times in } \tilde{\mathbb{L}} \end{cases}$$

Building on this definition, the authors introduced the concept of a three-subset division trail and detailed the propagation rules for the `COPY`, `XOR`, and `AND` operations in [33].

Proposition 2.12 (`COPY` [33]) *Let $F : \mathbb{F}_2 \to \mathbb{F}_2 \times \mathbb{F}_2$ such that $F(x) = (y_0, y_1)$ where $y_0 = y_1 = x$, and $\mathbf{u} \to (\mathbf{v}_0, \mathbf{v}_1)$ be a three-subset division trail w.r.t the function F. Then, the valid three-subset division trails satisfy*

$$\texttt{COPY}(\mathbf{u}, \mathbf{v}_0, \mathbf{v}_1) : \begin{cases} \mathbf{v}_0 + \mathbf{v}_1 & \geqslant \mathbf{u} \\ \mathbf{u} & \geqslant \mathbf{v}_0 \\ \mathbf{u} & \geqslant \mathbf{v}_1 \end{cases}$$

where $\mathbf{u}, \mathbf{v}_i$ are binary variables, for all $0 \leqslant i \leqslant 1$, representing the input and output three-subset division property without unknown subset w.r.t the `COPY` function.

Proposition 2.13 (`XOR` [33]) *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2$ such that $F(x_0, x_1, \ldots, x_{n-1}) = y$ where $y = x_0 \oplus \ldots \oplus x_{n-1}$, and $(\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{n-1}) \to \mathbf{v}$ be a three-subset division trail w.r.t the function F. Then, the valid CBDP trails satisfy*

$$\texttt{XOR}(\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{n-1}, \mathbf{v}) : \mathbf{v} - \mathbf{u}_0 - \mathbf{u}_1 - \ldots - \mathbf{u}_{n-1} = 0.$$

where $\mathbf{u}_i, \mathbf{v}$ are binary variables, for all $0 \leqslant i \leqslant (n-1)$, representing the input and output three-subset division property without unknown subset w.r.t the `XOR` function.

Proposition 2.14 (`AND` [33]) *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2$ such that $F(x_0, x_1, \ldots, x_{n-1}) = y$ where $y = \bigwedge_{i=0}^{n-1} x_i$, and $(\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{n-1}) \to \mathbf{v}$ be a three-subset division trail w.r.t the function F. Then, the valid three-subset division trail satisfy*

$$\texttt{AND}(\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{n-1}, \mathbf{v}) : \mathbf{v} = \mathbf{u}_i, \text{ for all } i \in \{0, 1, n-1\}.$$

where $\mathbf{u}_i, \mathbf{v}$ are binary variables, for all $0 \leqslant i \leqslant (n-1)$, representing the input and output three-subset division property without division property w.r.t the `AND` function.

Recovering ANF Coefficients for Public Functions: Algorithm Description [33].

Let f denote a Boolean function defined on an n-bit string $x = (x_0, x_2, \ldots, x_{n-1})$. This function is structured through iterative applications of fundamental public operations. The ANF of f is presented as follows.

$$f(x) = \bigoplus_{u \in \mathbb{F}_2^n} a_u^f x^u$$

We intend to determine a_u^f for a specific u. This requires constructing a MILP (Mixed Integer Linear Programming) model that embodies the three-subset division property without an unknown subset of the function f. In their work [33], the authors introduced Algorithm 1, aimed at recovering a coefficient a_u^f from the ANF of f. Initially, the three-subset division property without an unknown subset is defined by u, and the MILP solver enumerates feasible solutions. The performance of Algorithm 1 hinges on the quantity of these solutions.

Algorithm 1 Algorithm for retrieving an ANF coefficient [33]

Input: The vector v, the MILP model $\mathcal{M}$, represents the propagation of three-subset division property without unknown subset of the function f.
Output: The value of a_v^f.
Let $\mathtt{x}_i$ represent an MILP variable from $\mathcal{M}$ that corresponds to the i-th input of f
$\mathcal{M}.\mathtt{con} \leftarrow \mathtt{x}_i = 1$ for all indices i such that $v_i = 1$
$\mathcal{M}.\mathtt{con} \leftarrow \mathtt{x}_i = 0$ for all indices i such that $v_i = 0$
Execute the MILP model $\mathcal{M}$ and enumerate all possible solutions
if the count of feasible solutions is even **then**
    $a_v^f = 0$
else
    $a_v^f = 1$
return a_v^f

2.3 Overview of Cryptanalytic Techniques

Cryptanalysis of symmetric-key cryptosystems involves a diverse set of techniques, each exploiting different structural, statistical, or algebraic properties of a cipher. While these aforementioned techniques (differential cryptanalysis, impossible differential cryptanalysis, linear cryptanalysis, integral cryptanalysis, division property, and cube attack) have been developed independently, they can be classified into broader categories based on their underlying principles. In this section, we categorize, compare, and analyze the relevance of various cryptanalytic methods to provide a cohesive understanding of their applicability.

2.3.1 Classification of Cryptanalytic Techniques

To better structure the discussion, we classify the major cryptanalytic techniques into three broad categories:

1. **Statistical Attacks:** These methods exploit biases in the distribution of ciphertexts or intermediate values to distinguish a cipher from a random permutation or to recover the key.
    - Differential Cryptanalysis
    - Impossible Differential Cryptanalysis
    - Linear Cryptanalysis
2. **Structural Attacks:** These techniques focus on the internal structure of the cipher, such as its S-boxes, key schedule, or state propagation properties.
    - Integral Cryptanalysis
    - Division Property
3. **Algebraic Attacks:** These methods model the cipher using algebraic equations and attempt to solve them to extract key information.
    - Cube Attacks

Each of these categories serves a different purpose and has distinct strengths and limitations. In the following subsections, we provide a comparative discussion and highlight their relevance.

2.3.2 Comparative Analysis of Cryptanalytic Techniques

Each cryptanalytic technique is best suited to certain cipher structures and attack models:

- **Differential Cryptanalysis** is highly effective against Feistel networks and SPN structures, especially those with weak diffusion.
- **Impossible Differential Cryptanalysis** become useful when standard differential trails are too probable, and they work well on ciphers with long diffusion layers.
- **Linear Cryptanalysis** is often used when an S-box exhibits linear biases that can be exploited for key recovery.
- **Integral Cryptanalysis** is applicable when the cipher exhibits strong structural patterns that allow certain sums of state values to be predicted.
- **Division Property-based Attacks** extend integral cryptanalysis to cases where classical integral properties do not hold, making them useful for analyzing ciphers with complex linear layers.
- **Cube Attacks** leverage algebraic properties and are effective against ciphers with polynomial structure, particularly certain stream ciphers and lightweight block ciphers.

2.3.3 Summary

By structuring these techniques into broad categories and comparing their applications, we provide a more systematic understanding of symmetric-key cryptanalysis. The choice of technique depends on the target cipher's structure, the available attack model, and computational feasibility. The automated techniques developed in this thesis build upon these classical cryptanalytic methods, enabling efficient attack construction and security evaluation of modern symmetric-key primitives.

2.4 Cryptosystems Mentioned in the Thesis

In this thesis, we employ several cryptosystems as targets for attacks to highlight their vulnerabilities. This section provides descriptions of these cryptosystems.

2.4.1 SIMON

SIMON is a family of lightweight ciphers designed by the National Security Agency (NSA) in 2013 [14] to provide high security and efficiency for use in constrained environments. It is based on a typical Feistel design, where each block is divided into two halves and consists of bitwise AND, rotation, and XOR operations. SIMON has several variants on n-bit words and the block size is $2n$ bits for $n \in \{16, 24, 32, 48, 64\}$. The key size is a multiple of n by m, for $m \in \{2, 3, 4\}$. Each variant can be denoted as SIMON2n/mn. The number of rounds depends on the block size and key size, according to Table 2.1.

Variant	Block size $2n$	Key size mn	Word size n	Key words m	Round T
SIMON32	32	64	16	4	32
SIMON48	48	72	24	3	36
		96		4	36
SIMON64	64	96	32	3	42
		128		4	44
SIMON96	96	96	48	2	52
		144		3	54
SIMON128	128	128	64	2	68
		192		3	69
		256		4	72

Table 2.1: SIMON parameters

Figure 2.10 illustrates the operations of the round function. Let L_i, R_i represent the left and right n-bit input words to the i-th round of SIMON, the output of i-th round L_{i+1}, R_{i+1} is

computed as:

$$R_{i+1} = L_r$$
$$L_{i+1} = R_r \oplus K_i \oplus ((L_i \lll 8) \odot (L_i \lll 1)) \oplus (L_i \lll 2)$$

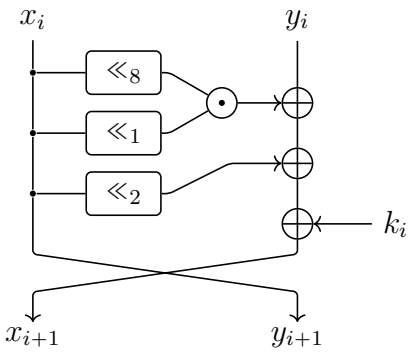


Figure 2.10: Round function of SIMON

The key schedule function of SIMON is linear. Depending on the size of the master key, the subkeys are derived from the mn-bit master key, which is divided into m words, each of n bits. A more detailed specification can be found in [14].

2.4.2 Simeck

Simeck is also a family of lightweight Feistel block ciphers introduced in CHES 2015 [19] that combines the most advantageous features from both SIMON and SPECK. The Simeck block cipher with a $2n$-bit block is denoted Simeck2n/4n, where $n \in \{16, 24, 32\}$, with a $4n$-bit key. More specifically, there are 3 variants of Simeck, namely Simeck32/64, Simeck48/96 and Simeck64/128. The corresponding rounds for these variants are 32, 36, and 44, respectively. As shown in Figure 2.11, the round function of Simeck is also composed of three operations: AND, rotation, and XOR, slightly modified from SIMON.

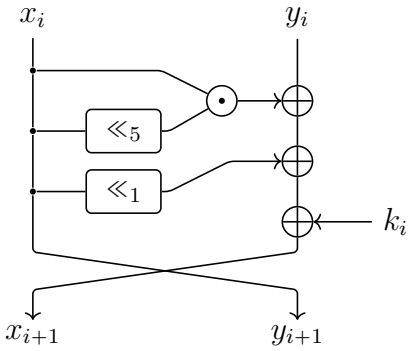


Figure 2.11: Round function of Simeck

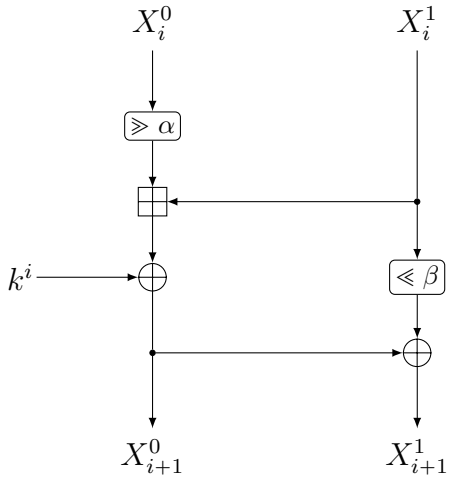


Figure 2.12: Round Function of SPECK

Let L_i, R_i represent the left and right n-bit input words to the i-th round of Simeck, the output (L_{i+1}, R_{i+1}) of the i-th round is computed as:

$$\begin{aligned} R_{i+1} &= L_r \\ L_{i+1} &= R_r \oplus K_i \oplus (L_i \odot (L_i \lll 1)) \oplus (L_i \lll 2) \end{aligned}$$

The key schedule of Simeck uses an LFSR procedure. A given master key generates the subkeys. For more details of Simeck key scheduling, we refer the reader to [19].

2.4.3 SPECK

The lightweight block cipher SPECK [14] was announced by the NSA in 2013. SPECK2n/mn has a block size of $2n$ bits and a key size of mn bits, where n can be 16, 24, 32, 48, and 64, and m can be 2, 3, or 4.

The round function of SPECK is defined in Figure 2.12 with the rotation parameter $(\alpha, \beta) = (7, 2)$ if the block size is 32, and $(8, 3)$ otherwise. Let (x_i^0, x_i^1) be the input of the i-th round and (x_{i+1}^0, x_{i+1}^1) be the output. In each round, the state is updated as follows:

$$(x_{i+1}^0, x_{i+1}^1) = R_{k^i}(x_i^0, x_i^1) = (((x_i^0 \gg \alpha) \boxplus x_i^1) \oplus k^i, (x_i^1 \lll \beta) \oplus (((x_i^0 \gg \alpha) \boxplus x_i^1) \oplus k^i))$$

where k^i is the round key. The key schedule reuses the round function to generate round keys.

2.4.4 LEA

LEA (Lightweight Encryption Algorithm) [18] is a block cipher developed by the Korea Internet & Security Agency (KISA) to provide lightweight encryption in resource-constrained environments. LEA encrypts data 1.5–2 times faster than AES, the most popular block cipher. Operating on 128-bit blocks, LEA supports key lengths of 128, 192, and 256 bits. The round's numbers are 24, 28, and 32 for 128-, 192- and 256-bit keys, respectively.

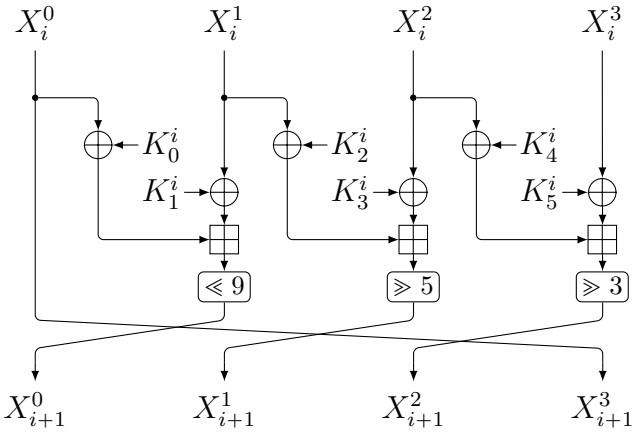


Figure 2.13: LEA

The encryption algorithm of LEA divides a plaintext of four 32-bit words $(x_0^0, x_0^1, x_0^2, x_0^3)$ into a ciphertext $(x_i^0, x_i^1, x_i^2, x_i^3)$, where r represents the number of rounds. The round function for round $r, r = 0, \ldots, n-1$ is defined as follows:

$$\begin{aligned}
x_{i+1}^0 &\leftarrow ((x_i^0 \oplus k_i^0) \boxplus (x_i^1 \oplus k_i^1)) \lll 9 \\
x_{i+1}^1 &\leftarrow ((x_i^1 \oplus k_i^2) \boxplus (x_i^2 \oplus k_i^3)) \gg 5 \\
x_{i+1}^2 &\leftarrow ((x_i^2 \oplus k_i^4) \boxplus (x_i^3 \oplus k_i^5)) \gg 3 \\
x_{i+1}^3 &\leftarrow x_i^0
\end{aligned}$$

where $k_i = k_i^0, k_i^1, k_i^2, k_i^3, k_i^4, k_i^5$ is the round key generated by the key schedule. One round of LEA can be seen in Figure 2.13.

2.4.5 ChaCha

Chacha [15] is a stream cipher designed by Daniel J. Bernstein. It belongs to the family of ciphers known as Salsa20. ChaCha operates on 512-bit blocks, which are divided into 16 words, and supports key lengths of 128 or 256 bits. The state of ChaCha can be presented as a 4×4 matrix:

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ t_0 & t_1 & v_0 & v_1 \end{pmatrix}$$

In the initial matrix, the first row contains four fixed constants: c_0, c_1, c_2, c_3. The following two rows contain eight key elements each: $k_0, k_1, \ldots, k_7$. The last row starts with the block counter and is followed by the nonce.

An operation called a quarter-round updates the rows and columns, transforming a four-

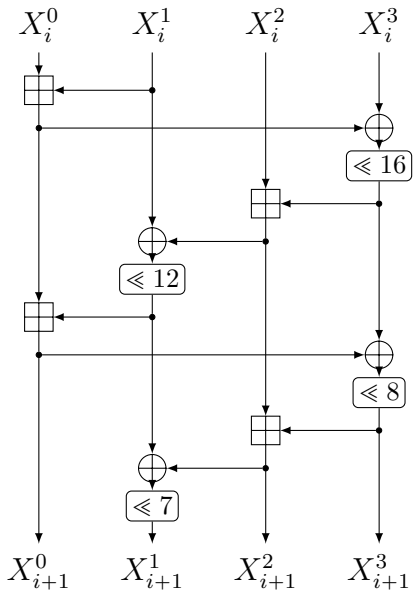


Figure 2.14: Quarter-round function of ChaCha

word vector (a, b, c, d) into (a'', b'', c'', d'') via an intermediate vector (a', b', c', d'):

$$
\begin{aligned}
a' &= a \boxplus b, \\
d' &= (d \oplus a') \lll 16, \\
c' &= c \boxplus d, \\
b' &= (b \oplus c') \lll 12, \\
a'' &= a' \boxplus b', \\
d'' &= (d' \oplus a'') \lll 8, \\
c'' &= c' \boxplus d'', \\
b'' &= (b' \oplus c'') \lll 7.
\end{aligned}
$$

Figure 2.14 illustrates the round function of ChaCha.

2.4.6 SipHash

SipHash is a family of pseudorandom functions introduced by Aumasson and Bernstein at Indocrypt 2012 [17], designed specifically for short message inputs. SipHash has an internal state size of 256 bits, uses a 128-bit key, and produces a 64-bit tag. SipHash variants are denoted as SipHash-c-d where c is the number of *Compression* rounds processing each message block and d is the number of *Finalization* rounds. The 64-bit tag is computed as follows:

- *Initialization:* Four 64-bit words of internal state v_0, v_1, v_2 and v_3 with the 128-bit key

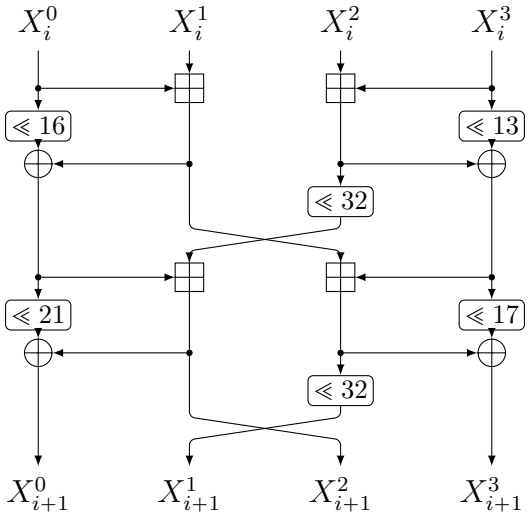


Figure 2.15: SipHash

$K = k_1||k_0$ are initialized as

$$
\begin{aligned}
v_0 &= k_0 \oplus 736f6d6570736575 \\
v_1 &= k_1 \oplus 646f72616e646f6d \\
v_2 &= k_0 \oplus 6c7967656e657261 \\
v_3 &= k_1 \oplus 7465646279746573
\end{aligned}
$$

- *Compression:* SipHash-c-d processes the b-byte string m by parsing it into 64-bit little-endian words. Each word is processed iteratively, first with $v_3 \oplus = m_i$, then through c iterations of SipRound, and finally with $v_0 \oplus = m_i$.
- *Finalization:* Once all message blocks are processed, the constant `ff` is xored with v_2. Then d iterations of SipRound are executed, and SipHash-c-d yields the 64-bit value: $v_0 \oplus v_1 \oplus v_2 \oplus v_3$

The round function of the SipHash is shown in Figure 2.15.

2.4.7 Chaskey

Chaskey is a permutation-based MAC algorithm presented by Mouha *et al.* in 2014 [16], inspired by Siphash. Chaskey processes an arbitrary-sized message M and a 128-bit key K. The message M is divided into blocks $m_1, m_2, \ldots, m_k$ of 128 bits each. In case the last block is incomplete, padding is applied. It generates a t-bit tag τ (where $t \leqslant n$) to authenticate the message M. The core function is a permutation constructed using the ARX design as depicted in Figure 2.16.

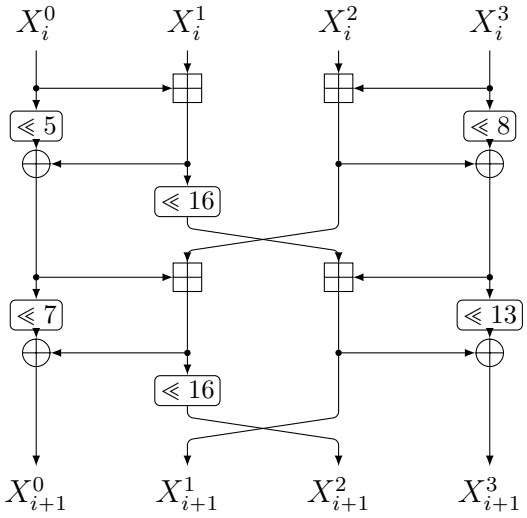


Figure 2.16: Chaskey

2.4.8 PRINCE

PRINCE [25] is a block cipher that operates on 64-bit blocks and uses a 128-bit key. This key is divided into two 64-bit segments, denoted as

$$k = k_0||k_1$$

and then expanded to 192 bits through the transformation

$$(k_0||k_1) \rightarrow (k_0||(k_0 \ggg 1) \oplus (k_0 \gg 63)||k_1) = (k_0||k_0'||k_1)$$

In this structure, the first two subkeys, k_0 and k_0', function as whitening keys, while k_1 serves as the 64-bit key for a 12-round block cipher which we will refer to as PRINCE$_{core}$. The whole encryption procedure of PRINCE$_{core}$ is illustrated in Figure 2.17.

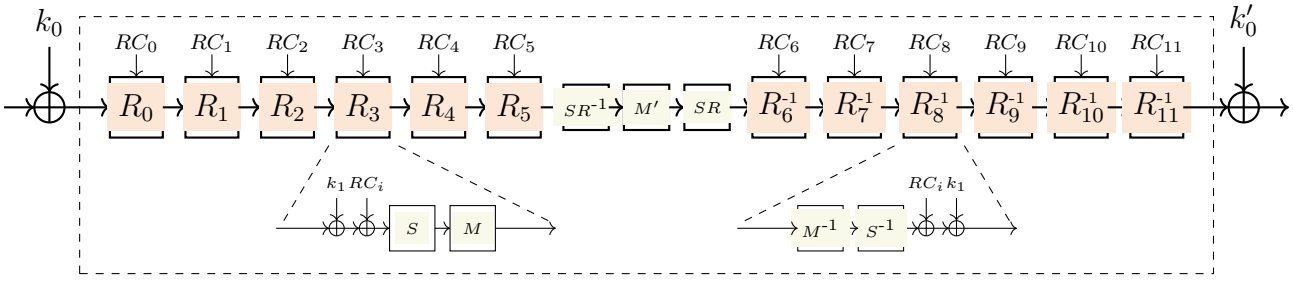


Figure 2.17: Structure of PRINCE cipher

In PRINCE$_{core}$, each round involves adding a secret key, applying S-boxes, performing a linear layer transformation, and adding a round constant. During this process, the 64-bit state undergoes an XOR operation with a 64 bit subkey. The cipher employs a single 4-bit S-box, as detailed in Table 2.2. In the Linear layer (also known as M, and M' layer), the 64-bit state is multiplied with a 64×64 matrix M (resp. M'). The M' layer is only used in

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

Table 2.2: PRINCE S-box

the middle round, and we construct the M layer by combining the M' layer with a Shiftrow (SR) operation as follows:

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

Additionally, we can write $M = SR \circ M'$. Now, we can use the following four 4×4 matrices as building blocks for the M' layer.

$$M_0 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, M_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

This yields $\hat{M}^0$ and $\hat{M}^1$, as shown below.

$$\hat{M}^0 = \begin{pmatrix} M_0 & M_1 & M_3 & M_4 \\ M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \end{pmatrix}, \hat{M}^1 = \begin{pmatrix} M_1 & M_2 & M_3 & M_0 \\ M_2 & M_3 & M_0 & M_1 \\ M_3 & M_0 & M_1 & M_2 \\ M_0 & M_1 & M_2 & M_3 \end{pmatrix}$$

Finally, we construct a 64×64 block diagonal matrix M' with

$$M' = (\hat{M}^0, \hat{M}^1, \hat{M}^1, \hat{M}^0)$$

For more details, please refer to [25].

2.4.9 MANTIS

Since the structures of the block ciphers MANTIS [26] and PRINCE are similar, the whitening keys are employed both preceding and following the main components. Using 64 bit subkeys k_0 and k_1, the 128-bit key is initially split into k_0, and k_1. Subsequently, after spliting the key is extended to the 192-bit key

$$(k_0||k_1) \rightarrow (k_0||(k_0 \gg 1) \oplus (k_0 \gg 63)||k_1) = (k_0||k_0'||k_1)$$

$m = m_0||m_1||\cdots||m_{15}$ is the plaintext that the cipher is given; the m_i are 4-bit cells. In addition, $IS_i = m_i$ is set for each $i \in \{0,1,\ldots,15\}$, which initiates the internal state of the cipher. An additional tweak input is sent to the cipher, which is $T = t_0||t_1||\ldots||t_{15}$.

Round Function. One round R_i of MANTIS operates on the cipher internal state depending on the round tweakey tk as

$$\texttt{MC} \circ \texttt{PC} \circ \texttt{AT} \circ \texttt{AC} \circ \texttt{SC}$$

where `MC`, `PC`, `AT`, `AC`, and `SC` are, respectively, mixcolumn, permutecell, addtweakey, addconstant, and subcell. The elements of the round function are explained as follows: The four-bit S-box used in MANTIS is the same as the MIDORI S-box $\texttt{Sb}_0$. The S-box is described as follows:

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\texttt{Sb}_0(x)$	c	a	d	3	e	b	f	7	8	9	1	5	0	2	4	6

Table 2.3: MANTIS S-box

The round constant RC_i undergoes a `XOR` operation with the state during each round, and the constants are determined in a manner similar to the method used in PRINCE. During round R_i, the complete round tweakey state $h^i(T) \oplus k_1$ is `XOR`ed with the cipher's internal state. Conversely, in the i-th inverse round, R_i^{-1}, the tweakey state $h^i(t) \oplus k_1 \oplus \alpha$ is `XOR`ed with the internal state, where $\alpha = \texttt{0x243f6a8885a308d3}$. The tweak permutation h is defined as follows:

$$h = (6, 5, 14, 15, 0, 1, 2, 3, 7, 12, 13, 4, 8, 9, 10, 11).$$

Additionally, the cells of the internal state are permuted according to the MIDORI permutation

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

Finally, each cipher internal state array column is multiplied by the binary matrix used in MIDORI and shown below.

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

For more details, please refer to [26].

2.4.10 KLEIN

KLEIN is a family of block ciphers [27] at RFIDSec2011. Due to its compact implementation and low memory requirements in both software and hardware, this cipher family is well-suited for resource-constrained devices like RFID tags and wireless sensors. Using 64-bit blocks, KLEIN functions as a Substitution-Permutation network (SPN). It is available in three variants: KLEIN-64, KLEIN-80, and KLEIN-96. These variants utilize key lengths of 64, 80, and 96 bits, respectively, and operate through 12, 16, and 20 rounds in their encryption process. In each round, four distinct layers are executed sequentially: `Add Roundkey`, `Sub Nibbles`, `Rotate Nibbles`, and `Mix Nibbles`. Initially, the state undergoes `Add Roundkey`, combined via `XOR` with the round key. Subsequently, the resulting output is segmented into

sixteen parts of 4 bits each, referred to as `nibbles`. Each nibble undergoes transformation through an identical involutive 4×4 S-box.

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

Table 2.4: KLEIN S-box

Then, `Rotate Nibbles` shifts the state by rotating two bytes to the left, and `Mix Nibbles` implements Rijndael `Mix column` transformation independently to different parts of the state. The resulting output comprises 4 bytes obtained through multiplication with a specified matrix.

$$M = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}$$

After the process, a final whitening key is added, which means the encryption routine uses one additional key compared to the number of rounds. Using an algorithm called `Key Schedule` that is based on a Feistel structure, the round keys are obtained from the `Master Key`. Please refer to [27] for more details.

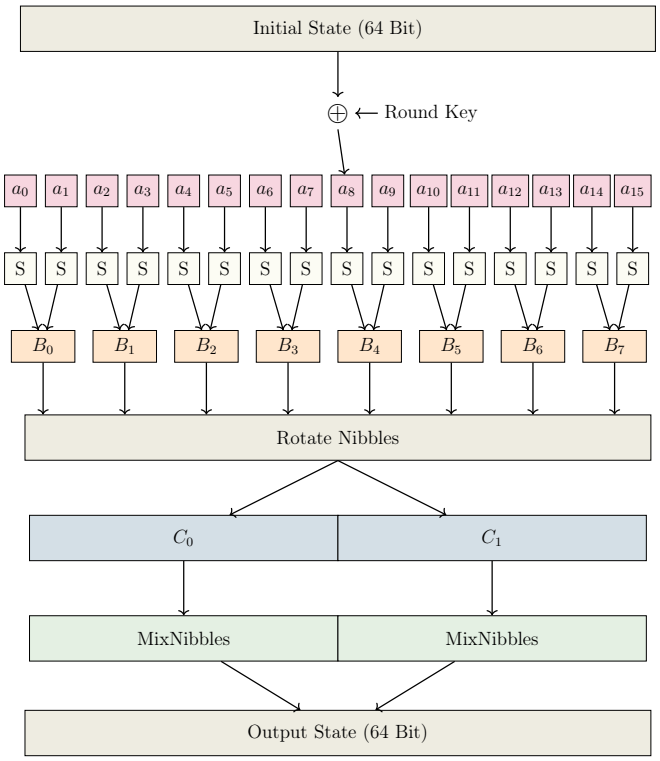


Figure 2.18: One-round structure of KLEIN

2.4.11 PRIDE

The 64-bit block cipher PRIDE uses a 128-bit key that adheres to an SPN structure. `Add Roundkey`, `Sub Cells`, and `Linear Layer` are the three operations that make up the round function. The cipher consists of 20 rounds, of which the first 19 are the same, and the final round's linear layer is skipped. The PRIDE S-box is as follows:

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

Table 2.5: S-box of PRIDE

The block cipher PRIDE incorporates a linear layer L that comprises a permutation P, a matrix M, and the inverse permutation P^{-1}. The matrix is M is formed by the product $L_0 \times L_1 \times L_2 \times L_3$. The permutation P of PRIDE as follows:

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$P(x)$	1	17	33	49	2	18	34	50	3	19	35	51	4	20	36	52
x	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
$P(x)$	5	21	37	53	6	22	38	54	7	23	39	55	8	24	40	56
x	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
$P(x)$	9	25	41	57	10	26	41	58	11	27	43	59	12	28	44	60
x	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
$P(x)$	13	29	45	61	14	30	46	62	15	31	47	63	16	32	38	64

Table 2.6: Permutation P of block cipher PRIDE

2.4.12 Grain-128AEAD

In the NIST LWC standardization process, Grain-128AEAD [32] is a finalist, and it incorporates several features initially introduced in its predecessor, Grain-128a [55]. However, there are four distinct differences between Grain-128AEAD and Grain-128a: increased MAC sizes, absence of a mode dedicated solely to encryption, enhanced security measures during initialization, and restrictions on keystream usage.

The internal state of Grain-128AEAD consists of two 128-bit registers denoted as b and s. The first register b is initialized with a 128-bit key k, while the second register s is initialized with a 96-bit initialization vector IV. The initial states are specifically detailed as follows:

$$\begin{cases} b = k \\ s = IV \parallel 1 \quad \text{(with all other bits set to 1 except for the final bit)} \end{cases}$$

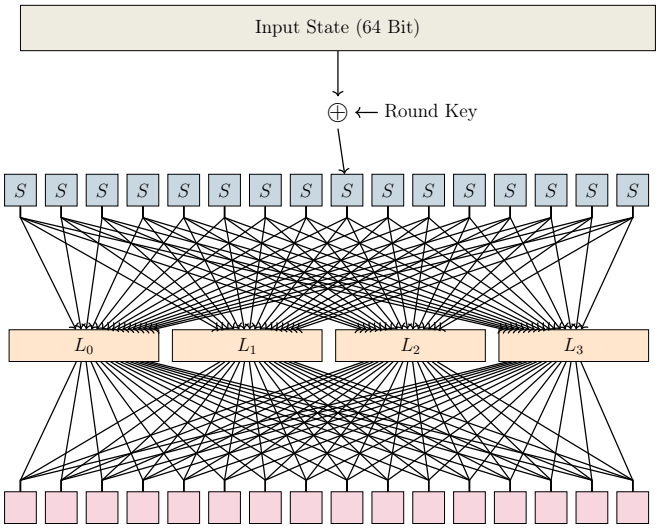


Figure 2.19: One-round structure of PRIDE

The set IV is represented by the notation IV_1, IV_2, ..., IV_{96}. Here is the pseudocode outlining the update procedure during the initialization stage.

$$\begin{cases} g \leftarrow b_0 + b_{26} + b_{56} + b_{91} + b_{96} + b_3b_{67} + b_{11}b_{13} + b_{17}b_{18} + b_{27}b_{59} \\ \qquad + b_{40}b_{48} + b_{61}b_{65} + b_{68}b_{84} + b_{88}b_{92}b_{93}b_{95} + b_{22}b_{24}b_{25} + b_{70}b_{78}b_{82}, \\ f \leftarrow s_0 + s_7 + s_{38} + s_{70} + s_{81} + s_{96}, \\ h \leftarrow b_{12}s_8 + s_{13}s_{20} + b_{95}s_{42} + s_{60}s_{79} + b_{12}b_{95}s_{94}, \\ z \leftarrow h + s_{93} + b_2 + b_{15} + b_{36} + b_{45} + b_{64} + b_{73} + b_{89}, \\ (b_0, b_1, \ldots, b_{127}) \leftarrow (b_1, \ldots, b_{127}, g + s_0 + z), \\ (s_0, s_1, \ldots, s_{127}) \leftarrow (s_1, \ldots, s_{127}, f + z). \end{cases}$$

The setup phase involves sequentially updating the state 256 times without generating any output. Following this, a change is implemented in the update mechanism to introduce an initial key stream output z. This adjustment presupposes that the first bit of the pre-output key stream becomes observable afterwards. These specifications outline how Grain-128AEAD differs from its predecessor and how its internal state is initialized and updated during setup.

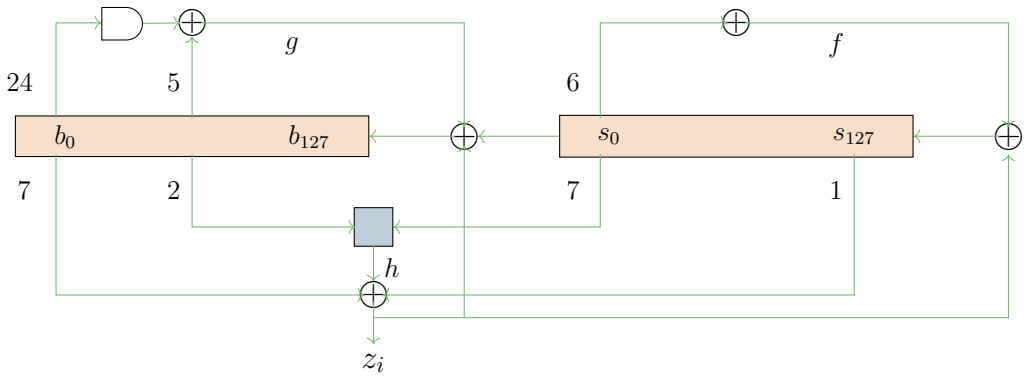


Figure 2.20: Structure of Grain-128AEAD

Chapter 3

Lower Bound on Number of Compression Calls of Collision-Resistance Preserving Hash

Hash functions serve as fundamental components in cryptography, and the task of designing a hash function that is both secure and efficient is a longstanding challenge in the field. Hash functions can be classified based on their internal construction, and while many hash functions are based on compression functions, there are also hash functions that do not rely on a distinct compression step, such as sponge functions [56] (these are a class of hash functions that absorb input data and then "squeeze" out the hash value), polyhash [57] (this is a polynomial evaluation-based hash function that operates by evaluating a polynomial over the input data), etc.

The process of creating a hash function based on a compression function can generally be broken down into two steps. The first step involves constructing a compression function, which takes inputs of a fixed length and generates smaller outputs than the inputs. The primary goal of a compression function is collision resistance, which means it should be computationally infeasible to find two different inputs that result in the same output.

Historically, block ciphers and permutations have been widely utilized in constructing compression functions [58]–[62]. Initially, block ciphers stood out as the preferred primitives for building compression functions, exemplified by the foundational designs of MD5, SHA1, and SHA2 hash functions. Notably, among the domain-extending algorithms, the Merkle-Damgård [63], [64] (MD) and Merkle tree constructions are the most prominent examples.

The Merkle-Damgård hash function technique involves creating collision-resistant cryptographic hash functions by utilizing collision-resistant compression functions. Merkle trees were introduced by Ralph Merkle in 1980 as a method for verifying large public files [65]. Since then, they have found wide-ranging applications in cryptography. These include but are not limited to parallel hashing, ensuring the integrity of large files, long-term data storage, various signature schemes [66]–[69], time-stamping mechanisms [70], protocols based on zero-knowledge proofs [71], [72], and even in the development of anonymous cryptocurrencies [73]. These applications demonstrate the versatility and importance of Merkle trees in modern cryptographic systems.

A collision-resistant hash function is one of the earliest primitives with plenty of crypto-

graphic applications, starting from digital signatures [74], [75], proof of membership (Merkle trees [64]), encryption, authentication, etc. The Merkle-Damgård construction gained popularity because Merkle and Damgård demonstrated that if the compression function used is resistant to collisions, the resulting hash function will also be collision-resistant. We can call this property *collision-resistance preserving* property. Merkle-Damgård and Merkle tree hash have the collision-resistance preserving property, i.e., if the compression function is collision-resistant, then so is the hash function constructed using it. The number of compression function calls for both constructions is the same. For example, we can process ℓ block messages by making ℓ or $(\ell-1)$ calls to the underlying $2n$-to-n bit compression function for both constructions. The question of "How can the Merkle-Damgård hash construction be improved by processing the same number of messages with fewer compression calls?" addresses an important aspect of cryptographic efficiency. Compression function calls are the core operations in hash constructions. Each call requires processing input blocks and applying cryptographic algorithms, which can be computationally expensive. By reducing the number of compression calls, the overall computational effort required to hash a message is minimized, leading to faster performance. Moreover, a hash mode that processes the same amount of data using fewer calls can achieve higher throughput, making it more suitable for applications requiring rapid data processing, such as secure communication protocols and blockchain systems.

Now, an improvement in the context of cryptographic hash constructions is not merely about reducing computational costs but achieving a better balance between efficiency and security. For a revised construction to qualify as an improvement the new construction must provide the same or better security guarantees as the original Merkle-Damgård and Merkle tree. In summary, an "improvement" in this sense ensures that the same cryptographic objectives are met with less computational effort, making the construction more appealing in terms of both performance and practicality while maintaining robust security guarantees. Hence, the question arises:

Can we improve Merkle-Damgård and Merkle tree constructions?

In a recent publication [8], the authors introduced a novel $5n$-to-n bit hash function denoted as T_5. This function utilizes three $2n$-to-n compression functions, surpassing the current standards set by Merkle-Damgård and Merkle trees. Therefore, T_5 construction is more efficient than the Merkle-Damgård construction when processing 5 message blocks because it requires only three compression function calls, compared to the four calls needed by Merkle-Damgård and Merkle Tree. Hence, T_5 achieves the same goal of processing the message while reducing the number of compression function calls, as fewer calls translate to lower computational costs, making T_5 more resource-efficient. Moreover, in [9], a perfect binary tree hash function (ABR) of height ℓ, which processes $(2^\ell + 2^{\ell-1} - 1)$ message blocks using $(2^\ell - 1)$ calls to underlying $2n$-to-n-bit compression function, was introduced. The collision security of T_5, and ABR hash function with height 3 (a hash function that processes 11 blocks of message using seven calls to $2n$-to-n bit compression functions) was proved under the ideal model assumption of the compression functions in [8], [10]. The authors in [9] claimed optimal birthday-bound collision resistance of ABR hash of height ℓ for any ℓ under the random oracle model. However, the proof is incorrect and reported [10], and the claim is still unproven.

In the ideal model, Stam [76] conjectured that for an ℓn-to-n bit hash function using r calls to tn-to-n bits compression functions, the minimum number of compression function calls required to achieve optimal birthday security is given by $r \geqslant (2\ell - 1)/(2t - 1)$. This bound is popularly known as Stam's bound, and it was later proven in two works by Steinberger [77] and by Steinberger, Sun and Yang [78]. In these works, the authors used an ideal oracle and proved that one can break the constructed primitive in unbounded time but with a bounded number of queries. Therefore, in the extensively researched scenario where $t = 2$[1], the established minimum value is $r \geqslant (2\ell - 1)/3$ as a lower limit. This results in a 1.5 times efficiency gap compared to the efficiency of Merkle-Damgård and Merkle tree.

Are we done?

Investigating the potential for the most optimal ℓn-to-n bit collision-resistant hash function becomes an exciting research direction, presuming the underlying compression functions possess collision-resistant properties. This examination aims to discern whether there are opportunities for enhancing the Merkle-Damgård and Merkle tree constructions within such a framework. We already know that for a general hash function based on a compression function, Steinberger, Sun, and Yang [77], [78] proved a lower bound on the number of compression function calls to achieve birthday-bound collision security in the random oracle model conjectured by Stam [76], which motivates us to propose some lower bound on the number of compression function calls to achieve collision-resistance preserving property for a general hash function. Hence, the following questions remain open.

Main Problem of the chapter: Can we propose any collision-resistance preserving hash function improving over the state-of-the-art Merkle-Damgård and Merkle-Tree hash? What about the lower bound on the number of compression function calls for collision-resistance preserving hash functions?

3.0.1 Our Contributions

To address the problem, we investigate the collision-resistance preserving properties of various hash modes such as T_5, ABR hash mode, and the extended version of Shrimpton-Stam Hash [79]. Our findings reveal that these modes do not have the collision-resistance preserving property. Given the significance of collision-resistance preserving properties in cryptographic hash modes, this chapter aims to establish a lower bound on the number of compression function calls required for collision-resistant hash modes, assuming that the underlying compression functions are collision-resistant.

Collision-Resistance Preserving Property: Lower Bound. The above results motivate us to think about the lower bound on the number of underlying compression function calls to achieve the collision-resistance preserving property of a hash mode. Analyzing the

[1]In many hash function designs, the compression function is a $2n$-to-n bit function ($t = 2$). This is because the compression function typically takes two blocks of input: one block representing the intermediate hash value (or chaining value) and the other block representing the current message segment. This structure is a standard design choice in hash modes like Merkle-Damgård. Cryptographic hash functions such as SHA-1, SHA-256, and others based on the Merkle-Damgård construction use $2n$-to-n bit compression functions. Analyzing $t = 2$ directly provides insights into the security and efficiency of these widely deployed algorithms.

structures of various hash modes, including the well-known Merkle-Damgård and Merkle tree constructions, as well as the recently proposed T_5 and ABR constructions, we observe that all functions, apart from the underlying compression functions, are linear in these hash modes (e.g., Figure 3.6, Figure 3.2). We refer to this category of hash modes as *linear hash modes*. A detailed description is provided in Subsection 3.2.3. Notably, to the best of our knowledge, all existing hash modes fall into the category of linear hash modes. Consequently, our initial focus is determining a lower bound on the number of compression function calls required to achieve the collision-resistance preserving property in a linear hash mode. In particular, we prove the following statement:

Let H be an ℓn-to-n bit linear hash mode using r many tn-to-n bit compression function calls, satisfies either

$$(i)\ r < \lceil (\ell-1)/(t-1) \rceil,\ \textit{if}\ t \geqslant 2$$
$$\textit{or}$$
$$(ii)\ \ell \geqslant 2,\ \ \textit{if}\ t = 1$$

with $(tr + \ell + r) < 2^n$. Then H is not collision-resistant, only assuming that the underlying compression functions are collision-resistant.

Our objective is to demonstrate that a hash mode H, which is an ℓn-to-n bit linear hash mode utilizing r calls to tn-to-n bit compression function does not possess the collision-resistance preserving property under specific conditions. Specifically, if $r < \lceil (\ell-1)/(t-1) \rceil$, if $t \geqslant 2$ or $\ell \geqslant 2$, if $t = 1$, and $(tr + \ell + r) < 2^n$, then H fails to be collision-resistance preserving. To prove this, it suffices to construct a collision-resistant compression function f and show that H^f is not collision-resistant[2]. Specifically, we establish that for a linear hash mode H meeting the specified conditions, a particular type of collision-resistant compression function f exists such that H^f is not collision-resistant. Constructing such an f requires the *assumption* that collision-resistant hashes exist. We state and prove Theorem 3.2 in Section 3.3.

As we already know, Merkle-Damgård and Merkle tree hash functions process ℓ block messages using $\lceil (\ell-1)/(t-1) \rceil$ many tn-to-n bit compression function calls with $t \geqslant 2$, and they have collision-resistance preserving property, which implies that we cannot improve the construction of Merkle-Damgård and Merkle tree hash function in light of the collision-resistance preserving property. Hence, these two constructions are optimum while considering the class of linear hash mode.

As of the second contribution of our chapter, we also prove that if an ℓn-to-sn-bit linear hash mode is designed using r many tn-to-n-bit compression function calls with $(tr+\ell+r) < 2^n$, and $t, s \geqslant 2$ then $r \geqslant \lceil (\ell-s)/(t-1) \rceil$ to achieve collision-resistance preserving property. More precisely, we prove the following statement:

Let H be an ℓn-to-sn bit linear hash mode using r many tn-to-n bit compression function calls, satisfies $r < \lceil (\ell-s)/(t-1) \rceil$, with $t, s \geqslant 2$, and $(tr + \ell + r) < 2^n$. Then H does not have collision-resistance preserving property.

[2]It is important to note that a collision-resistance preserving mode must ensure a secure hash for all collision-resistant compression functions (e.g., Merkle-Damgård hash), not just random ones.

We state and prove Theorem 3.7 in Section 3.5. It is important to note that all existing hash modes are linear, which is best known to us. This indicates that the linear hash modes represent a substantial class of hash functions, reflecting the importance of our results. Although we can consider simple non-linear functions apart from the underlying compression function, we do not know how to get a lower bound on those constructions, which looks pretty hard.

Organization of the Chapter. In Section 3.1, we provide a mathematical background of hash function and collision security (both in uniform and non-uniform setup). We also describe white-box and black-box reductions of collision security for hash modes. In Section 3.2, we describe the linear hash mode and its security. In Section 3.3, we state and prove our main result Section 3.2. In Section 3.4, we provide proof of Lemma 3.3, which is used in proving our main theorem. Finally, in Section 3.5, we extend our lower bound results for linear hash modes with more than one output block.

3.1 Background

In this chapter, $n \in \mathbb{N}$ is considered to be a security parameter. We call the elements of $\{0,1\}^n$ blocks. For any nonempty subset $\mathcal{L} \subseteq \mathbb{N}$, we write $\{0,1\}^{\mathcal{L}\cdot n} = \bigcup_{l\in\mathcal{L}}\{0,1\}^{ln}$[3]. Let $\mathbb{F} := \mathbb{F}_{2^n}$ denote the Galois field over $\{0,1\}^n$ with bitwise addition $a+b$ and field multiplication $a \cdot b$. We write $\mathbf{0} := 0^n$ and $\bar{\mathbf{1}} = 0^{n-1}1$ (the additive and multiplicative identities of the field, respectively). For a statement $P(m)$ we write $\forall^* m\ P(m)$ if there exists a positive integer M such that for all integers $m > M$, $P(m)$ is true (the notion $\forall^* m$ represents for all sufficiently large m). In other words, $P(m)$ is true for all sufficiently large m. A non-negative function $\epsilon(\cdot)$ is called negligible if $\forall k$, $\forall^* m$, $\epsilon(m) \leqslant m^{-k}$.

Complexity Model. We fix a reasonable computational model (polynomial equivalent to the Turing machine), and the runtime of all algorithms is computed under that model. The runtime also includes the size of the algorithm's description, which would help to avoid storing arbitrarily large advise strings implicitly. It also includes reading the input and writing its output. Any algorithm A has an input set of the form $\cup_{n\in\mathbb{N}}(\{1^n\} \times D_n)$ for some $D_n \subseteq \{0,1\}^*$. The run time of A is said to be $t(n)$ if the runtime for computing $A(1^n, x)$ is at most $t(n)$ for all $x \in D_n$. If $t(n)$ is a polynomial, we call the algorithm A and the function $A(\cdot)$ realized by the algorithm *polynomial time computable* (or simply "efficient").

3.1.1 Hash Function

A most general hash function is defined over $\{0,1\}^*$. In the first step, an appropriate padding rule is applied to the message so that the size of the padded message is a multiple of n. Then, the hash computation is applied to the padded message with the help of some other building blocks. For the sake of simplicity, we ignore the padding rule and restrict the domain of

[3]Precisely, $\{0,1\}^n$ represents one block, and $\{0,1\}^{\ell n}$ represents ℓ many blocks, where ℓ is an integer. However, instead of ℓ, it could be a set $\mathcal{L}$ which is a subset of $\mathbb{N}$. In this case, each element in $\mathcal{L}$ depicts the possible number of input blocks. For example, if $\mathcal{L} = \{2\}$, then $\{0,1\}^{\mathcal{L}\cdot n}$ means there will be two input blocks and n represents the size of each block.

our hash functions to $\{0,1\}^{\mathcal{L}\cdot n}$ for an appropriate set $\mathcal{L}$. We also restrict the hash output to n-bit or a multiple of n-bits.[4] An algorithm or function A is called (D,R)-function if for all $x \in D$, $A(x) \in R$.

Definition 3.1 *An $\mathcal{L}$-to-s computation F (or simply c-to-s computation when $\mathcal{L} = \{c\}$) is an efficiently computable function F such that for all $n \in \mathbb{N}$, $F_n := F(1^n, \cdot)$ is a $(\{0,1\}^{\mathcal{L}\cdot n}, \{0,1\}^{s\cdot n})$-function.*

Generalized Hash Mode. A hash mode is a computation that uses some building blocks (a relatively smaller domain) as oracles. A tuple of functions $H = (g_1, \ldots, g_r, g)$ is called (ℓ, r, t)-*mode* if

- g, called a *final output function*, is a $(\ell + r)$-to-1 computation, and
- g_i's, called *intermediate processing functions*, are $(\ell + i - 1)$-to-t computations, $i \in [r]$.

An (ℓ, r, t)-mode induces an ℓ-to-1 computation by applying r executions of t-to-1 computations as oracles. More formally, given r many t-to-1 computation oracles $\mathcal{O}_1, \ldots, \mathcal{O}_r$, we define the ℓ-to-1 computation $H^{\mathcal{O}_1,\ldots,\mathcal{O}_r}$ as

$$H^{\mathcal{O}_1,\ldots,\mathcal{O}_r}(1^n, M) = g(1^n, x_1, \ldots, x_{\ell+r})$$

where $M = (x_1, \ldots, x_\ell) \in \mathbb{F}^\ell$ and for $i \in [r]$, $\overrightarrow{y^{(i)}} \in \mathbb{F}^t, x_{i+\ell} \in \mathbb{F}$ (called *intermediate chaining inputs and outputs*, respectively) are calculated as follows (see Figure 3.1):

for $i = 1$ to r
- $\overrightarrow{y^{(i)}} = g_i(1^n, M, x_{\ell+1}, \ldots, x_{\ell+i-1})$;
- $x_{i+\ell} = \mathcal{O}_i(1^n, \overrightarrow{y^{(i)}})$;

We also write the intermediate chaining outputs as $\text{OUT}_H^{\mathcal{O}_1,\ldots,\mathcal{O}_r}(M) = (x_1, \ldots, x_{\ell+r})$ (whenever understood, we skip the notation H). For notational simplicity, we write $\mathcal{O}^r$ to denote r-tuple of oracles $(\mathcal{O}_1, \ldots, \mathcal{O}_r)$ and we skip 1^n (whenever n is understood or fixed in the context).

For fixed t-to-1 functions $f_1, \ldots, f_r$, $H^{f_1,\ldots,f_r}$ is a ℓ-to-1 hash function. A hash mode essentially transforms several small domain hash functions into a larger domain hash function (provided ℓ is larger than t).

3.1.2 Collision Security

Uniform vs. Non-uniform Collision Finder. We call A uniform collision-finder (or simply collision finder) if for all n, it returns a pair (M, M') of polynomial-bounded size. In notation, $(M, M') \leftarrow A(1^n)$. Let $\alpha_n \in D_n$ be a fixed string for each n. We call A non-uniform collision finder with an advise string $(\alpha_n)_n$ if for all n, $(M, M') \leftarrow A(1^n, \alpha_n)$. We also denote a non-uniform algorithm by a pair $(A, (\alpha_n))$. When α_n is constant for all n, it becomes

[4] One can similarly consider other sizes of hash outputs and a similar analysis of our chapter would follow.

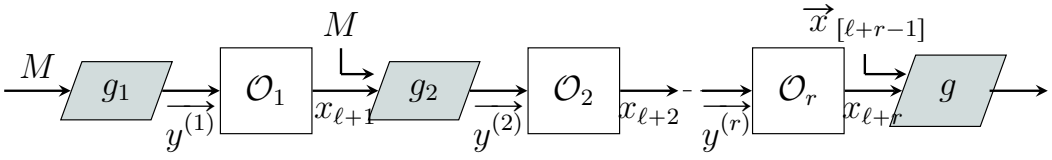


Figure 3.1: General Hash mode with ℓ message blocks based on r calls to underlying t-to-1 oracles.

a uniform collision finder, and we skip the notation (α_n) because it can be hard-coded in constant size inside the description of A.

Collision Security of a Hash Function. In the following we adopt the definition from [80]. A hash computation or hash function is also a computation whose collision security is considered. For an $\mathcal{L}$-to-s hash computation H, we define the *collision advantage of A* (uniform) as

$$\mathbf{CP}_{H,n}(A) = \Pr[(M, M') \leftarrow A(1^n), H(1^n, M) = H(1^n, M')].$$

We similarly define the collision advantage for non-uniform collision finders as

$$\mathbf{CP}_{H,n}(A, (\alpha_n)_n) = \Pr[(M, M') \leftarrow A(1^n, \alpha_n), H(1^n, M) = H(1^n, M')].$$

We say that $(A, \alpha := (\alpha_n)_n)$ (or simply A in the case of the uniform algorithm) is a successful collision finder of H if A runs in polynomial time and $\mathbf{CP}_{H,n}(A, \alpha)$ is non-negligible. We write

$$\mathbf{CP}_{H,n}(t, \alpha) = \max_A \mathbf{CP}_{H,n}(A, \alpha), \quad \mathbf{CP}_{H,n}(t) = \max_A \mathbf{CP}_{H,n}(A)$$

where the maximum is taken over all $t(n)$-time collision finder A.

Definition 3.2 (Collision-Resistant Hash Function) *A hash function H is called $(t(n), \epsilon(n), \alpha)$ **non-uniform collision-resistant** if $\boldsymbol{CP}_{H,n}(t(n), \alpha) \leqslant \epsilon(n)$ $\forall^* n$. We call H collision-resistant against an advise string (α_n) if for all polynomial $t(n)$, $\boldsymbol{CP}_{H,n}(t, \alpha)$ is negligible. (In other words, there is no efficient successful (non-uniform with the advise string α) collision finder of H.) When α_n is constant we call H **uniform collision-resistant** and we skip the notation α_n.*

Collision Security Model and Assumption. Informally, a hash function H is called *collision-resistant* if it is *hard* to find a collision pair (M, M') of H (i.e., $M \neq M'$ and $H(M) = H(M')$). It is easy to see there is no meaningful way to formalize the notion of collision resistance for a single hash function. Every function H with domain size larger than range (we call those functions compression functions) must be non-injective; hence, a collision pair exists. So, a short and fast program that outputs (M, M'), which are hardwired in the program, finds a collision pair. In other words, an efficient collision finding algorithm always exists, even if we currently may not know how to write it down. To overcome this issue, one may consider a family of hash functions $H = \{H_K : K \in \mathcal{K}\}$, and the definition says that when a uniformly random public key is used, no adversary can find a collision with

non-negligible probability. Another possibility is to consider a sequence of hash functions written as $H(1^n, \cdot)$, which is associated with each choice of a security parameter n, and so there exists collision-resistant hash function against a uniform adversary as one cannot simply hardwire all collision pairs into a uniform adversary for all n. So, we must assume existence of collision resistant hash function against uniform adversaries.

Classical Collision-Resistance Assumption. There exists a 2-to-1 collision-resistant function against all uniform collision finder algorithms. Using known collision-resistance preserving modes such as MD Hash, we can also assume a collision-resistant c-to-1 compression function H exists for every $c \geqslant 2$. We prove our main result (Theorem 3.2) in the uniform setting under this classical collision-resistance assumption. Then, we discuss our result in the non-uniform setting (Theorem 3.5) as well.

Note that for any hash function H (so a collision pair (M_n, M'_n) exists for all n), a non-uniform algorithm based on the advise string $(M_n, M'_n)_n$ can return a collision pair for all n. In other words, a collision secure hash function cannot exist against all non-uniform collision finders. However, till now, any universal advise string (α^*_n) is known such that for every H there exists A (depending on H) so that $A(1^n, \alpha^*_n)$ finds a collision pair for H for all n. This motivates us to pose the following assumption (a general assumption than the classical assumption).

Collision-Resistance Assumption against Non-uniform Collision Finder. For every advise string (α_n), a collision-resistant 2-to-1 compression function H against (α_n) exists. When α_n is constant, this is the same as the classical collision-resistance assumption for uniform adversaries.

Comparison of our model with Random oracle model. In [8], and [10], the authors provided collision security bound in a random oracle model (i.e., the underlying compression functions f of hash mode T_5 and ABR are assumed to be ideal, which means the compression functions act like a perfect random function). For this kind of model, the assumption is that a random oracle exists, which is definitely a strong assumption. In this context, our security assumption is that a collision-resistant compression function exists against different kinds of adversaries (for example, uniform adversaries), which is a practically more feasible assumption compared to random oracle model assumption.

3.1.3 Reduction of Collision Security

In [80], the author explained that, if a cryptographic protocol Π employs a hash function (keyed or unkeyed) H, then to prove the security of Π using a reduction-based approach, one can demonstrate the statement in existential form (C0): *If there is an effective algorithm A for attacking protocol Π, then there's an effective algorithm B for finding collisions in H.* For the unkeyed hash function H, this statement is trivially true. Therefore, the author in [80] stated constructive reductions defined as follows:

Code-Constructive Form (C1) [80]: *If you know an effective algorithm A for attacking protocol Π then you know an effective algorithm B for finding collisions in H.*

Black-Box-Constructive Form (C2) [80]: *If you possess effective means A to attack the protocol Π, then you have effective means B to find collisions in H.*

Black-box and White-Box Reduction for Collision Security. Based on our current knowledge, it is infeasible to prove unconditionally that a hash function is collision-resistant or to prove the collision-resistant assumptions. However, we can prove a reduction that shows collision security of H given that H' is collision-resistant, where H' is used to define H. In particular, we consider two types of reduction, namely black-box reduction (C2 form) and white-box reduction (C1 form) as defined below.

Definition 3.3 (Black-box and White-box Reduction) *Let H, H' be some hash functions and α, α' be some advise strings.*

1. *We call collision security of H is α* black-box reduced to $H'_1, \ldots, H'_r$ *if there is an efficient algorithm A such that for any sequence of collision pairs (M_n, M'_n) of $H(1^n, \cdot)$, $A(1^n, (\alpha_n, M_n, M'_n)_n)$ is a successful collision finder of $H'_i(1^n, \cdot)$, for some $i \in [r]$.*

2. *We call collision security of H is (α, α')* white-box reduced to $H'_1, \ldots, H'_r$ *if there is an efficient algorithm A such that for any successful collision finder $A'(1^n, \alpha')$ of $H(1^n, \cdot)$, $A(1^n, (\textsc{code}(A'), \alpha_n, \alpha'_n)_n)$ is also a successful collision finder of $H'_i(1^n, \cdot)$ for some $i \in [r]$, where $\textsc{code}(A')$ represents the algorithmic description of A'.*

We skip the notations α and α' whenever these are constant.

A black-box reduction algorithm (C2) finds a collision of H whenever a collision pair of H' is given to it. Whereas, a white-box reduction (C1) algorithm finds a collision of H whenever a code of an efficient collision-finder of H' is given. So, the existence of black-box reduction implies the existence of white-box reduction. For example, it is very well known that the Merkle-Damgård hash function, Merkle tree hash function, etc., are collision-resistant given that the underlying compression function is collision-resistant. This is essentially a black-box reduction. Those reductions can be easily converted to white-box also.

Remark: 1 *According to the general definition of black-box reduction, a hash function H is considered black-box reduced to H' if an efficient algorithm B can attack the protocol H, enabling the construction of an algorithm A to find collisions in H'. In this specific context of collision, we simplify the definition of black-box reduction. If there is an efficient algorithm B to find collisions in H, B simply returns the collision pairs for H. The efficient algorithm A then uses those collision pairs from H (provided by algorithm B) to construct collision pairs for H'.*

3.1.4 Input-Output Fixing Collision-Resistant Compression Function

An *input-output fixing function* f is a function that maps a fixed set of inputs to a fixed set of outputs. Let $\alpha = (\alpha_1, \ldots, \alpha_p) \in D^p$ and $\beta = (\beta_1, \ldots, \beta_p) \in R^p$, consisting of distinct values for some fixed p. A function $f : D \to R$ is called $\alpha \mapsto \beta$ input-output fixing mapping if for all i, $f(\alpha_i) = \beta_i$. We now extend this definition to an asymptotic setup. Let $\alpha = (\alpha_n)_n$ and $\beta = (\beta_n)_n$ where $\alpha_n = (\alpha_{n,1}, \ldots, \alpha_{n,p}) \in (\{0,1\}^{nt})^p$ and $\beta_n = (\beta_{n,1}, \ldots, \beta_{n,p}) \in (\{0,1\}^n)^p$ consist of distinct values for all n. We call any such pair (α, β) (t, p) in-out pair. A (t, p) in-out pair is called *computable* if there is an efficient algorithm G such that $G(1^n)$ returns (α_n, β_n) for all n.

Definition 3.4 (Input-Output Fixing Function) *Let (α, β) be a (t, p) in-out pair. A t-to-1 function f is called an $\alpha \mapsto \beta$ input-output fixing mapping, if*

$$f(1^n, \alpha_{n,i}) = \beta_{n,i}, \text{ for all } i \in [p], n \in \mathbb{N}.$$

In other words, for all n, the function $f_n := f(1^n, \cdot)$ is $\alpha_n \mapsto \beta_n$ input-output fixing mapping.

Proposition 3.1 *Let (α, β) be a (t, p) in-out computable pair. There is a $\alpha \mapsto \beta$ input-output fixing collision-resistant compression function provided the classical collision-resistant assumption is true.*

Proof. Let $c \geqslant 2t$ be some fixed integer. Suppose h' is a c-to-1 collision-resistant compression function (it exists by our classical collision-resistant assumption). For all sufficiently large n, we define $k := \lfloor n/2 \rfloor > \lceil \log p \rceil$ and $n' = n - k$. Suppose $q \in \{0,1\}^k$ is different from the last k bits of all $\beta_{n,i}$'s and be the smallest possible value among all k-bit values when considered as an integer (note that it can be computed efficiently as we can compute α_n and β_n efficiently). Now we define

$$h_{\alpha,\beta}(1^n, x) = \begin{cases} \beta_{n,i} & \text{if } x = \alpha_{n,i}, \text{ for all } i \in [p] \\ h'(1^{n'},\ x \parallel 0^{n(c-t)-ck}) \parallel q & \text{otherwise} \end{cases} \tag{3.1}$$

Clearly, $h_{\alpha,\beta}$ is a $\alpha \mapsto \beta$ input-output fixing mapping. Now, we prove that $h_{\alpha,\beta}$ is collision-resistant. The function $h_{\alpha,\beta}$ is easily seen to be efficiently computable.

We now provide a black-box reduction. Let an adversary A on the collision resistance of $h_{\alpha,\beta}$ be given, which returns a pair (M, M'). Now, whenever (M, M') is a collision pair, then both M, and M' can not be $\alpha_{n,i}$, $\forall i \in [p]$ as α_n, and β_n are two tuples of distinct values, for all n. Moreover, it can not be possible that M is one of $\alpha_{n,i}$ for some $i \in [p]$, and M' is other than $\alpha_{n,j}$, $\forall j \in [p]$ as q is different from the last k bits of all $\beta_{n,i}$'s. Therefore, $(M \parallel 0^{n(c-t)-ck}, M' \parallel 0^{n(c-t)-ck})$ is a collision pair of h'_{n-k}. ▫

3.2 Linear Hash Modes

3.2.1 Linear Algebra Lemma

Vectors. Given two m-dimensional vectors $\overrightarrow{u} = (u_1, \ldots, u_m), \overrightarrow{v} = (v_1, \ldots, v_m) \in \mathbb{F}^m$, we compute the dot-product

$$\overrightarrow{u} \cdot \overrightarrow{v} := \sum_{i=1}^{m} u_i \cdot v_i.$$

Given a set of m-dimensional vectors $\mathcal{V}$, we denote the following terminologies widely used in linear algebra:

- $\mathsf{span}(\mathcal{V})$: the set of all vectors which are linearly dependent on the vectors of $\mathcal{V}$;
- $\mathsf{null}(\mathcal{V})$: the set of all m-dimensional vectors $\overrightarrow{x}$ such that $\overrightarrow{v} \cdot \overrightarrow{x} = \mathbf{0}$ for all $\overrightarrow{v} \in \mathcal{V}$;

- $\mathsf{rank}(\mathcal{V})$: the maximal number of linearly independent vectors in $\mathcal{V}$;

We note that $\mathsf{rank}(\mathsf{span}(\mathcal{V})) = \mathsf{rank}(\mathcal{V})$. The *rank-nullity theorem* says that

$$\mathsf{rank}(\mathcal{V}) + \mathsf{rank}(\mathsf{null}(\mathcal{V})) = m.$$

A function $f : \mathbb{F}^m \to \mathbb{F}^t$ is called m-to-t function.

Lemma 3.1 (Linear Algebra Lemma) *Let $\mathcal{S}, \mathcal{N} \subseteq \mathbb{F}^m \backslash \{\mathbf{0}^m\}$ be two nonempty sets such that (i) $\mathcal{N}$ and $\mathsf{span}(\mathcal{S})$ are disjoint and (ii) $|\mathcal{N}| < 2^n$. Then,*

$$\textit{there exists } \overrightarrow{y} \in \mathsf{null}(\mathcal{S}) \textit{ such that } \forall \overrightarrow{v} \in \mathcal{N},\ \overrightarrow{v} \cdot \overrightarrow{y} \neq \mathbf{0}.$$

Equivalently, $\exists \overrightarrow{y} \in \mathsf{null}(\mathcal{S}) \backslash \bigcup_{\overrightarrow{v} \in \mathcal{N}} \mathsf{null}(\mathcal{S} \cup \{\overrightarrow{v}\})$.

*Proof.*Note that for all $\overrightarrow{v} \in \mathcal{N}$, $rank(\mathcal{S} \cup \{\overrightarrow{v}\}) = r + 1 \leqslant m$ where $r = rank(\mathcal{S})$. So, by rank-nullity theorem $|\mathsf{null}(\mathcal{S})| = (2^n)^{m-r}$ and $|\mathsf{null}(\mathcal{S} \cup \{\overrightarrow{v}\})| = (2^n)^{m-r-1}$ for all $v \in \mathcal{N}$. Hence,

$$|\mathsf{null}(\mathcal{S}) \backslash \bigcup_{\overrightarrow{v} \in \mathcal{N}} \mathsf{null}(\mathcal{S} \cup \{\overrightarrow{v}\})| \geqslant 2^{nm-nr} - |\mathcal{N}| 2^{nm-nr-n} \geqslant 2^{nm-nr} \left[1 - \frac{|\mathcal{N}|}{2^n}\right] > 0.$$

This shows that the above set is nonempty and hence the proof follows. □

Remark: 2 *Given the set of vectors $\mathcal{S}, \mathcal{N}$, in order to find such*

$$\overrightarrow{y} \in \mathsf{null}(\mathcal{S}) \backslash \bigcup_{\overrightarrow{v} \in \mathcal{N}} \mathsf{null}(\mathcal{S} \cup \{\overrightarrow{v}\}),$$

it is sufficient to find a basis of $\mathsf{null}(\mathcal{S})$, and $\mathsf{null}(\mathcal{S} \cap \{\overrightarrow{v}\})$ for all $\overrightarrow{v} \in \mathcal{N}$. A basis of the null space of a set of vectors may be computed by Gaussian elimination or may be computed by the Bareiss algorithm [81], which may work more efficiently than Gaussian elimination. The efficiency of this algorithm depends on the cardinalities of $\mathcal{S}$ and $\mathcal{N}$.[5]

Let $\overrightarrow{u} \in \mathbb{F}^m$, $\overrightarrow{v} \in \mathbb{F}^{m'}$ and $\mathcal{Z} \subseteq [m]$. We write $\overrightarrow{u}|_{\mathcal{Z}} = (u'_1, \ldots, u'_m)$ where $u'_i = u_i$ for all $i \notin \mathcal{Z}$, otherwise $u'_i = \mathbf{0}$. We call $\overrightarrow{u}|_{\mathcal{Z}}$, *$\mathcal{Z}$-projected vector* of $\overrightarrow{u}$. Note that

$$\text{for all } \overrightarrow{x} \in \mathbb{F}^m, \quad \overrightarrow{u}|_{\mathcal{Z}} \cdot \overrightarrow{x} = \overrightarrow{u_{\mathcal{Z}}} \cdot \overrightarrow{x_{\mathcal{Z}}}.$$

[5] In our main proof, where we use the results of Lemma 3.1, the cardinality of $\mathcal{S}$ and $\mathcal{N}$ is upper bounded by approximately $tr + \ell + r$, where the linear hash mode processes ℓ blocks of messages using r many tn-to-n bit compression function calls.

3.2.2 Linear Functions and Tuples of Vectors

m-to-1 Linear Function. To each vector $\overrightarrow{u} \in \mathbb{F}^m$, we associate a m-to-1 linear function u, defined as $u(\overrightarrow{x}) = \overrightarrow{u} \cdot \overrightarrow{x}$. For $i \in [m]$, the linear function $e_i(x_1, x_2, \ldots, x_m) = x_i$ has an equivalent vector representation $\overrightarrow{e_i}$, called the i^{th} *coordinate vector*, whose i^{th} element is $\mathbf{1}$, and all the remaining elements are $\mathbf{0}$. It is well known that every linear function over $\mathbb{F}$ can be uniquely represented by a function $u(\cdot)$ as defined above for some $\overrightarrow{u} \in \mathbb{F}^m$. So, we use the convention $\overrightarrow{u}$ to denote the corresponding vector for a linear function u and vice-versa. For $u : \mathbb{F}^\ell \to \mathbb{F}$ and $v : \mathbb{F}^r \to \mathbb{F}$, we define the joint linear function

$$(u \parallel v) : \mathbb{F}^{\ell+r} \to \mathbb{F}, \quad \overrightarrow{x} \mapsto u(x_1, \ldots, x_\ell) + v(x_{\ell+1}, \ldots, x_{\ell+r}), \ \overrightarrow{x} \in \mathbb{F}^{\ell+r}.$$

Note that $\overrightarrow{u \parallel v} = \overrightarrow{u} \parallel \overrightarrow{v}$, hence the concatenation notation $|$ is consistent in both vector and linear function representation.

m-to-t Linear Function. An m-to-t linear function $\mathbf{u} = (u_1, \ldots, u_t)$, can be represented by a t-tuple of vectors $\overrightarrow{\mathbf{u}} := (\overrightarrow{u_1}, \ldots, \overrightarrow{u_t}) \in (\mathbb{F}^m)^t$ such that for $\overrightarrow{x}$, we have

$$\mathbf{u}(\overrightarrow{x}) = (\overrightarrow{u_1} \cdot \overrightarrow{x}, \ldots, \overrightarrow{u_t} \cdot \overrightarrow{x}) := (u_1(\overrightarrow{x}), \ldots, u_t(\overrightarrow{x})).$$

Note that a pair of vectors $(\overrightarrow{u_1}, \overrightarrow{u_2})$ is different from the concatenation $\overrightarrow{u_1} \parallel \overrightarrow{u_2}$ as they correspond to two different linear functions.

For $k \in [t]$, we write $\overrightarrow{u_k} = (u_{k,1}, \ldots, u_{k,m}) \in \mathbb{F}^m$ (indices corresponding to components of vectors appear at the end). We use bold letters like $\mathbf{u}$ to denote m-to-t linear functions and vector arrow over bold letters like $\overrightarrow{\mathbf{u}}$ to denote t-tuples of m-dimensional vectors.

Tuple of m-to-t Linear Function. Now, we further generalize and consider a r-tuple of m-to-t linear function $\mathcal{U} := \left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}\right)$, i.e., for $i \in [r]$, $\mathbf{u}^{(i)}$ is a m-to-t linear function. This actually induces a linear function, abusing notation, $\mathcal{U} : \mathbb{F}^m \to (\mathbb{F}^t)^r$, such that for all $\overrightarrow{x} \in \mathbb{F}^m$,

$$\mathcal{U}(\overrightarrow{x}) = (\mathbf{u}^{(1)}(\overrightarrow{x}), \ldots, \mathbf{u}^{(r)}(\overrightarrow{x})).$$

We use calligraphic fonts to denote such tuples of m-to-t linear functions. We write

$$\overrightarrow{\mathbf{u}^{(i)}} := (\overrightarrow{u_1^{(i)}}, \ldots, \overrightarrow{u_t^{(i)}}), \quad \overrightarrow{u_j^{(i)}} := (u_{j,1}^{(i)}, \ldots, u_{j,m}^{(i)}) \in \mathbb{F}^m, i \in [r], j \in [t].$$

Definition 3.5 (Triangular-Dependent) *An m-to-t linear function $\mathbf{u}$ and its vector representation $\overrightarrow{\mathbf{u}} = (\overrightarrow{u_1}, \ldots, \overrightarrow{u_t})$ are called i-onward independent if $u_{k,i} = \ldots = u_{k,m} = \mathbf{0}$, for all $k \in [t]$.*

Let $r < m$. A r-tuple of m-to-t linear functions $\mathcal{U} := \left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}\right)$ is called triangular-dependent *if for all $i \in [r]$, $\mathbf{u}^{(i)}$ is $(i + (m - r))$-onward independent. We call $\mathcal{U}$ (ℓ, r, t)-triangular-dependent where $\ell = m - r$.*

If $\mathbf{u}$ is a m-to-t i-onward independent linear function then for all $\overrightarrow{x}, \overrightarrow{y} \in \mathbb{F}^m$ with $x_1 = y_1, \ldots, x_{i-1} = y_{i-1}$, we have $\mathbf{u}(\overrightarrow{x}) = \mathbf{u}(\overrightarrow{y})$. In other words, $\mathbf{u}(x_1, \ldots, x_m)$ is functionally independent of $x_i, \ldots, x_m$. This justifies the term "i-onward independent". So, for $1 \leqslant i \leqslant m$,

any $(i-1)$-to-t linear function $\mathbf{u}'$ is equivalent to a m-to-t i-onward independent linear function $\mathbf{u}$ such that

$$\mathbf{u}(x_1, \ldots, x_m) = \mathbf{u}'(x_1, \ldots, x_{i-1}), \quad \forall \overrightarrow{x} \in \mathbb{F}^m.$$

CONVENTION. Suppose $\mathbf{u}'^{(i)}$ is $(\ell+i-1)$-to-t linear function, $i \in [r]$. As discussed above, we can equivalently represent $\mathbf{u}'^{(i)}$ by i-onward independent m-to-t $\mathbf{u}^{(i)}$ linear function. Hence, we equivalently represent $\mathcal{U}' := \left(\mathbf{u}'^{(1)}, \ldots, \mathbf{u}'^{(r)}\right)$ by a triangular dependent r-tuple of m-to-t linear functions $\mathcal{U} := \left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}\right)$. We follow this convention while we define linear hash mode.

3.2.3 Linear Hash Mode

(ℓ, r, t)-Linear Hash Mode. *Linear Hash Mode* is a generalized hash mode $(g_1, \ldots, g_r, g)$ where the intermediate processing functions $g_i(1^n, \cdot)$'s and the final output function $g(1^n, \cdot)$ are linear for all n. We equivalently represent a linear hash mode by a pair $H(1^n, \cdot) := (\mathcal{U}_n := (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}), g(1^n, \cdot))$ where $\mathcal{U}_n$ is a triangular dependent r-tuple of m-to-t linear functions and g is a m-to-1 linear function, where $m = \ell + r$. We sometimes skip the notation 1^n for the sake of simplicity.

1. We call H *simple* if $g(x_1, \ldots, x_m) = x_m$. A simple linear hash mode is defined by $(\mathcal{U}_n := (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)}))_n$ and denoted as $\text{SLH}_{\mathcal{U}}$.

2. On the other hand, we call H *non-simple linear hash mode* if

$$g(x_1, \ldots, x_m) = c_1 x_1 + c_2 x_2 + \ldots + c_m x_m$$

where $c_m \neq \mathbf{0}$, and there exists at least one $i \in [m-1]$ such that $c_i \neq \mathbf{0}$. A non-simple linear hash mode is defined by $(\mathcal{U}_n, g)_n$ and is denoted as $\text{CLH}_{(\mathcal{U},g)}$.

Example 1 (Merkle-Damgård Hash Function) *Let $f : \mathbb{F}^t \to \mathbb{F}$ be a t-to-1 compression function. The Merkle-Damgård (MD) hash function derived from f, denoted $(MD)^f$ is a hash function defined over $\mathbb{F}^\ell$ using $r = \lceil (\ell-1)/(t-1) \rceil$, t-to-1 compression function calls that works as follows:*

$$(MD)^f(m_0, m_1, \ldots, m_{\ell-1}) = f\big(\cdots f(f(m_0, m_1, \ldots, m_{t-1}), m_t, \ldots, m_{2t-2}), \cdots, m_{\ell-1}\big)$$

Here, $m_i \in \mathbb{F}$, for all $0 \leqslant i \leqslant \ell - 1$. Merkle-Damgård hash is simple linear hash mode.

Example 2 (Merkle Tree Hash Function) *Merkle-tree variant for a binary tree (based on a 2-to-1 compression function) is a hash function using $(\ell-1)$ compression function calls where $\ell = 2^d$ for $d \in \mathbb{N}$. Like Merkle-Damgård, Merkle tree hash is also simple linear hash mode.*

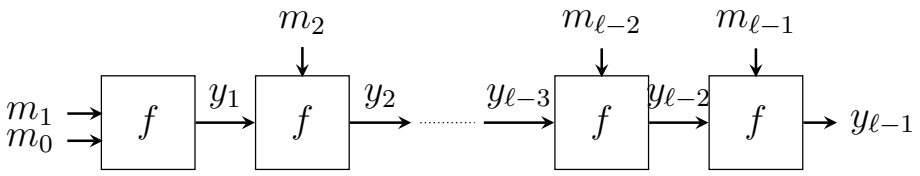


Figure 3.2: Merkle-Damgård hash mode processing ℓ block messages with $(\ell - 1)$ 2-to-1 compression function calls

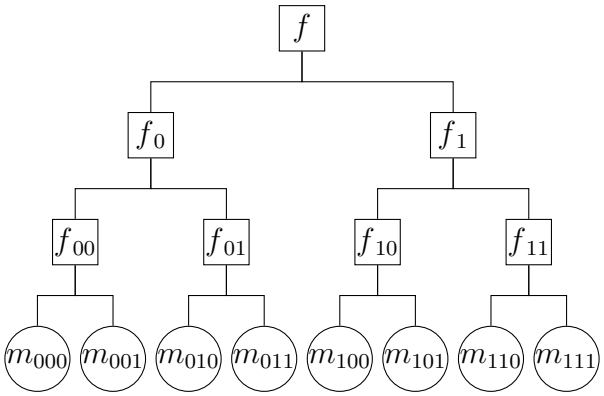
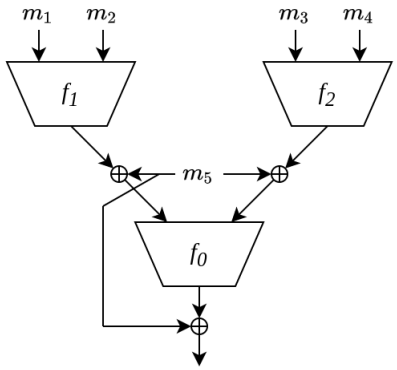
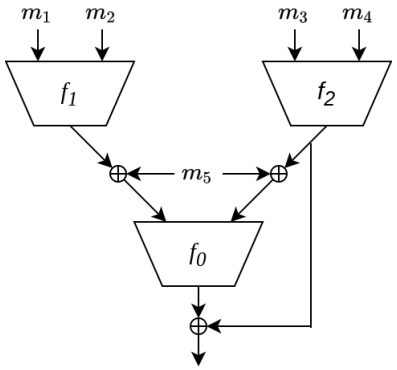


Figure 3.3: Merkle tree hash mode of height 3 processing 8 block messages with 7 2-to-1 compression function calls



(a) T_5 hash mode with 5 message blocks and 3 compression function calls.



(b) ABR hash mode of height 2 with 5 message blocks and 3 compression function calls.

Figure 3.4: The structures of T_5 and ABR hash modes.

Example 3 (T_5 & ABR Hash Function) *T_5 hash function, described in [8], is designed to handle a message input consisting of five blocks, which requires three 2-to-1 compression function calls. As discussed in [9], the ABR hash function operates as a perfect binary tree hash with a height denoted by ℓ. It processes a total of $m = (2^\ell + 2^{\ell-1} - 1)$ message blocks and executes $(2^\ell - 1) = (m - 2^{\ell-1})$ 2-to-1 compression function calls. It is important to highlight that both are non-simple linear hash modes.*

Example 4 (Shrimptom-Stam Hash Function) *Shrimpton-Stam construction [79] is a*

2n-to-n bit compression function based on three n-to-n-bit non-compressing primitives. Now, using three 2n-to-n-bit compression functions, we can compress five message blocks, adding one block for each compression function (trivially extended version[6] of Shrimptom-Stam construction). It's worth noting that Shrimpton-Stam hash mode falls under the category of non-simple linear hash modes.

3.3 Our Main Result

In this section, we present our main result, which establishes the lower bound on the number of compression function calls required to achieve collision-resistance preserving property in an (ℓ, r, t)-linear hash mode.

Theorem 3.2 ***[Main Theorem]*** *Let H be an (ℓ, r, t) linear hash mode satisfying either (i) $t \geqslant 2$, $r < \lceil(\ell-1)/(t-1)\rceil$, or (ii) $t = 1$, $\ell \geqslant 2$ with $(tr + \ell + r) < 2^n$. Then, there exists*

- *t-to-1 collision-resistant compression functions $f_1(1^n, \cdot), \ldots, f_r(1^n, \cdot)$[7] and*
- *an uniform collision finder A such that $\boldsymbol{CP}_{H^{f_1,f_2,\ldots,f_r},n}(A) = 1, \forall^* n$.*

In other words, the white-box (and hence black-box) reduction from H to $(f_1, \ldots, f_r)$ does not exist.

We already know (ℓ, r, t) linear hash mode having Merkle-Damgård and Merkle tree hash structures are collision-resistant assuming underlying compression functions are collision-resistant, and in this case $r = \lceil(\ell-1)/(t-1)\rceil$, for $t \geqslant 2$[8]. The above result says we cannot have a more efficient collision-resistance preserving hash mode than Merkle-Damgård and Merkle tree hash mode. Recently, T_5, ABR hash were shown to be collision-resistant in the random oracle model [8], [9]. The above theorem shows that it is impossible to prove the collision security of ABR and T_5 based on only the collision security assumption of the underlying compression function. We present the detailed attack on T_5 in Subsection 3.3.1. For the detailed attack on ABR hash mode, please refer to Subsection 3.3.2. It is important to note that Shrimpton-Stam construction also does not have collision-resistance preserving property. Similar reasoning can be applied as we do for T_5 constructions.

[6]Note that this construction has not been proposed formally.

[7]The phrase "there exists t-to-1 collision-resistant compression functions" refers to a specific category of collision-resistant compression functions for which we can prove that H is not collision-resistant, rather than any arbitrary collision-resistant compression function. To demonstrate the existence of this particular type of collision-resistant compression function, we must rely on the assumption that collision-resistant hashes exist.

[8]It is possible to design a linear hash mode H that processes ℓ block messages under the following conditions: either H uses $r > \lceil(\ell-1)/(t-1)\rceil$ t-to-1 compression function calls ($t \geqslant 2$), or $\ell < 2$ with t-to-1 compression function calls ($t = 1$), ensuring that H remains collision-resistant, provided the underlying hash function is also collision-resistant. For instance, consider a linear hash mode H that processes 5 message blocks using 5 2-to-1 compression function calls, defined as

$$H(m_0, \ldots, m_4) = f(MD(m_0, \ldots, m_4), m_0),$$

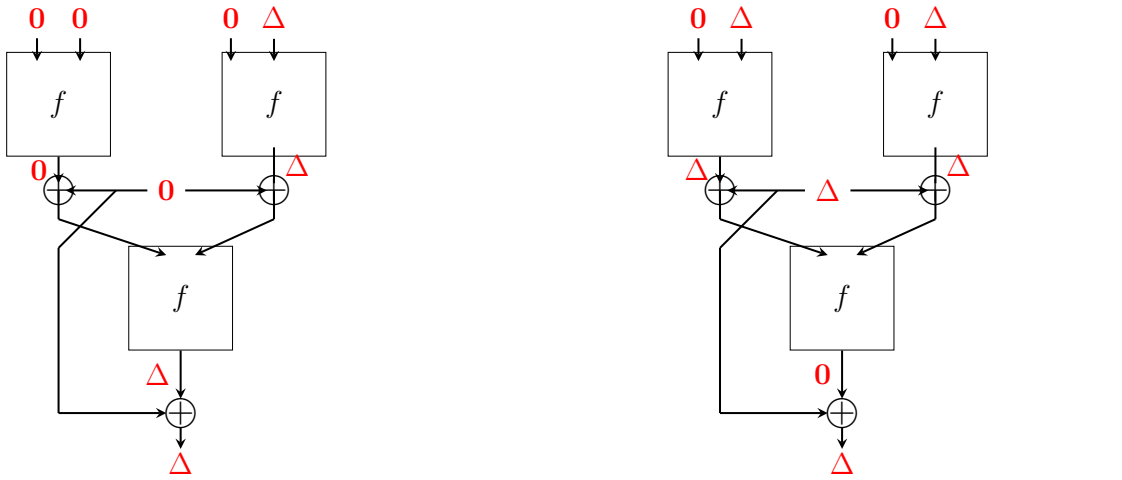
where MD is the Merkle-Damgård hash mode processing 5 message blocks with 4 compression function calls, and f is a 2-to-1 compression function. It can be shown that if the underlying compression function is collision-resistant, H is also collision-resistant.

3.3.1 Absence of Collision-Resistance Preserving Property in T_5

Our goal is to prove that T_5 does not have collision-resistance preserving property, i.e., there exists some collision-resistant compression function f, such that T_5^f is not collision-resistant. We start our proof by establishing an observation on T_5 hash mode. We notice that, if the underlying compression function f of the T_5 hash mode satisfy the following constraints:

$$f(\mathbf{0}, \Delta) = \Delta, \;\; f(\mathbf{0}, \mathbf{0}) = \mathbf{0} \tag{3.2}$$

then, two messages $(\mathbf{0}, \mathbf{0}, \mathbf{0}, \Delta, \mathbf{0})$, and $(\mathbf{0}, \Delta, \mathbf{0}, \Delta, \Delta)$ with non-zero message difference $(\mathbf{0}, \Delta, \mathbf{0}, \mathbf{0}, \Delta)$ construct a collision pair for T_5^f (see Figure 3.5). It is important to note that the conditions written in Equation 3.2 are not sufficient to conclude the collision-resistance property of the underlying compression function, but we need the underlying compression function to have the collision-resistance property (i.e., there may or may not exist collision-resistant compression function satisfying Equation 3.2, but ultimately we need the underlying compression function have to be collision-resistant and satisfy Equation 3.2). Hence, we have to design a collision-resistant compression function, say F, which satisfies Equation 3.2, and identify a message pair that produces a collision in the T_5^F hash mode.



(a) Processing of message $m = (\mathbf{0}, \mathbf{0}, \mathbf{0}, \Delta, \mathbf{0})$ through T_5^f

(b) Processing of message $m' = (\mathbf{0}, \Delta, \mathbf{0}, \Delta, \Delta)$ through T_5^f

Figure 3.5: Processing of non-zero message difference $(\mathbf{0}, \Delta, \mathbf{0}, \mathbf{0}, \Delta)$ through T_5^f, where f satisfies Equation 3.2

Based on this observation, we define a 2-to-1 compression function F as follows:

$$F(x) = \begin{cases} \mathbf{0} & \text{if } \; x = (\mathbf{0}, \mathbf{0}) \\ \Delta & \text{if } \; x = (\mathbf{0}, \Delta) \\ f'(x \parallel 0^{n-3k}) \parallel q & \text{Otherwise} \end{cases} \tag{3.3}$$

where f' is a 3-to-1 collision-resistant compression function (it exists by our classical collision-resistant assumption) and $q \in \{0,1\}^k \backslash \{0^k\}$ is different from the last k bits of Δ, and is the

smallest possible value among all k-bit values when considered as an integer with $k := \lceil n/2 \rceil$. It is noteworthy that, in Equation 3.3, f' takes an input of $3n'$ bits and generates an output of n' bits, where $n' = n - k$.

For instance, F can also be seen as $h_{\alpha,\beta}$ defined as in Equation 3.1 for $\alpha = (\alpha_1, \alpha_2) \in \{0,1\}^{4n}$ such that $\alpha_1 = (\mathbf{0}, \mathbf{0})$, and $\alpha_2 = (\mathbf{0}, \Delta)$, and $\beta = (\beta_1, \beta_2) = (\mathbf{0}, \Delta) \in \{0,1\}^{2n}$, with $p = t = 2$, $c = 3$. Therefore, using Proposition 3.1, we can easily conclude that F is collision-resistant. Finally, for the T_5 hash construction using F as its compression function, i.e., for T_5^F, if one process $M = (\mathbf{0}, \mathbf{0}, \mathbf{0}, \Delta, \mathbf{0})$ and $M' = (\mathbf{0}, \Delta, \mathbf{0}, \Delta, \Delta)$ in T_5^F, then

$$\mathrm{T}_5^F(M) = \mathrm{T}_5^F(M') = \Delta \neq \mathbf{0}.$$

Remark: 3 *To prove the general lower bound on linear hash mode H to achieve collision-resistance preserving property, mentioned in Section 3.0.1, we have to construct some particular type of collision-resistant compression function f, a similar construction is necessary as described in T_5. Hence, we establish such construction generally in Equation 3.1 (Subsection 3.1.4) and also prove that such construction is collision-resistant in Proposition 3.1. This construction we call an input-output fixing compression function (Definition 3.4).*

3.3.2 Absence of Collision-Resistance Preserving Property in General ABR Hash

ABR [9] mode of height ℓ with 2^ℓ leaf message inputs, with $r = 2^\ell - 1$ compression function calls and a total of $(2^\ell + 2^{\ell-1} - 1)$ input blocks. Therefore, we represent the input message blocks as $m_1, m_2, \ldots, m_{2^\ell + 2^{\ell-1} - 1}$, while the compression functions are denoted as $f_{i,j}$. Here, i ranges from 1 to ℓ, and for a given value of i, j takes on values from 1 to $2^{\ell-i}$ ($\ell = 3$ case is illustrated in Figure 3.6). Here, we would like to prove that the general ABR hash does not have black-box reduction.

In order to do this, first, we prove that ABR_2 does not have black-box reduction. Similarly, in ABR_2 hash mode, we notice that two different messages with non-zero message difference say $(\mathbf{0}, \mathbf{0}, \mathbf{0}, \Delta, \mathbf{0})$ would be a collision pair, provided the underlying compression function f satisfy certain input-output patterns ($f(\mathbf{0}, \Delta) = \Delta$, $f(\mathbf{0}, \mathbf{0}) = \mathbf{0}$) for some $\Delta \neq \mathbf{0}$. Therefore, if we process $M = (\mathbf{0}, \Delta, \mathbf{0}, \mathbf{0}, \Delta)$, and $M' = (\mathbf{0}, \Delta, \mathbf{0}, \Delta, \Delta)$ in ABR_2^F where 2-to-1 collision-resistant compression function F is defined as in Equation 3.3), then $\mathrm{ABR}_2^F(M) = \mathrm{ABR}_2^F(M') = \Delta$. Hence, we conclude that ABR_2 does not have black-box reduction.
Therefore, we can extend this attack to ABR hash mode of height ℓ for any integer $\ell \geqslant 3$. Now, the general ABR hash mode of height ℓ using F as its compression function calls where 2-to-1 collision-resistant compression function F is defined in Equation 3.3, can be written as

$$ABR_\ell^F(m_1, m_2, \ldots, m_{2^\ell + 2^{\ell-1} - 1}) = H(ABR_2^F(m_1, m_2, m_3, m_4, m_{2^\ell + 1}), m_5, \ldots, m_{2^\ell + 2^{\ell-1} - 1}) \tag{3.4}$$

where H is a hash function that takes an input of $(2^\ell + 2^{\ell-1} - 5)$ blocks and outputs one block, and it uses F as its compression function calls ($\ell = 3$ case is illustrated in Figure 3.6, where the dotted line implies the function ABR_2). Specifically, H contains all the other functions of ABR_ℓ^F except the leftmost ABR_2^F. If we consider the case of $\ell = 3$, H is a hash

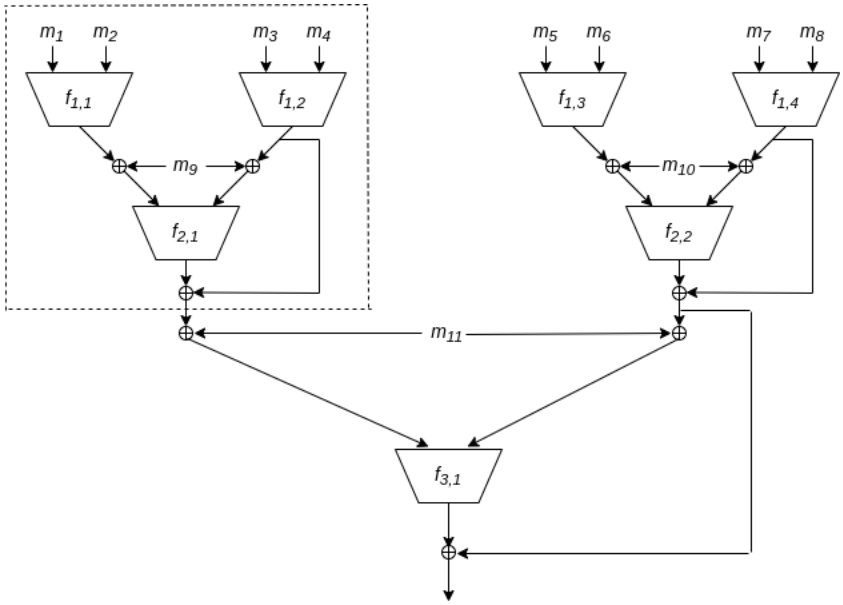


Figure 3.6: ABR mode of height 3 with 11 message blocks based on 7 calls to underlying 2-to-1 compression functions.

function which includes all the functions of ABR_3^F except the dotted part in Figure 3.6. This implies H takes input as $ABR_2^f(m_1, m_2, m_3, m_4, m_9), m_5, m_6, m_7, m_8, m_{10}, m_{11}$, and outputs $h = ABR_3^f(m_1, \ldots, m_{11})$.

Now, as we know, (M, M') is a collision pair of ABR_2^F as defined above, then we can construct a pair $(\bar{M}, \bar{M}')$ defined as follows:

$$\begin{cases} \bar{m_j} = m_j, \ \bar{m'_j} = m'_j & for\ j = 1,\ 2,\ 3,\ 4,\ 2^\ell + 1 \\ \bar{m_j} = \bar{m'_j} = \mathbf{0} & otherwise \end{cases} \tag{3.5}$$

Therefore, from Equation 3.4, using Equation 3.5 we can conclude that $(\bar{M}, \bar{M}')$ is a collision pair of ABR_ℓ^F which implies the general ABR hash mode of height ℓ does not have collision-resistance preserving property.

Remark: 4 *As elaborated in Example 4, the extended version of the Shrimption-Stam construction involves the processing of five message blocks through three calls to $2n$-to-n bit compression function. It is crucial to recognize that demonstrating collision resistance for this construction based solely on the collision-resistant property of the underlying compression function is unattainable. Similar reasoning can be applied as we do when establishing the impossibility of collision reduction for T_5 and ABR constructions.*

3.3.3 Proof of Theorem 3.2

In this section, we demonstrate the proof of Theorem 3.2. First, we state and explain some necessary definitions and results to finally prove the Theorem 3.2.

DIFFERENCE PROPAGATION THROUGH (ℓ, r, t) LINEAR HASH MODE. Let H be an (ℓ, r, t) linear hash mode where $\mathcal{U} := (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)})$. Suppose we process two messages $M = (x_1, \ldots, x_\ell) \in \mathbb{F}^\ell$, and $M' = (x'_1, \ldots, x'_\ell) \in \mathbb{F}^\ell$ through $H^{\mathcal{O}^r}$ and let the intermediate chaining outputs and inputs be

$$\begin{aligned} \text{OUT}_H^{\mathcal{O}^r}(M) &= (x_1, \ldots, x_m), \;\; \mathcal{U}(\overrightarrow{x}) = \mathcal{Y} := (\overrightarrow{y^{(1)}}, \ldots, \overrightarrow{y^{(r)}}) \\ \text{OUT}_H^{\mathcal{O}^r}(M') &= (x'_1, \ldots, x'_m), \;\; \mathcal{U}(\overrightarrow{x'}) = \mathcal{Y}' := (\overrightarrow{y'^{(1)}}, \ldots, \overrightarrow{y'^{(r)}}) \end{aligned}$$

Let $m = \ell + r$, $\delta x_i = x_i \oplus x'_i$ for all $i \in [m]$. Clearly,

$$\mathbf{u}^{(i)}(\delta x_1, \ldots, \delta x_m) = \delta\overrightarrow{y^{(i)}} := \overrightarrow{y^{(i)}} \oplus \overrightarrow{y'^{(i)}}.$$

Note that

$$\delta\overrightarrow{y^{(i)}} = \mathbf{0}^t \Rightarrow \delta x_{i+\ell} = \mathbf{0}, \quad \forall i \in [r]$$

(zero differences in inputs of the oracles would lead to zero difference in the output). However, to avoid collisions in the inputs and outputs of the oracles, we also need non-zero differences in the inputs to lead to non-zero differences in the outputs, and hence, we require

$$\delta\overrightarrow{y^{(i)}} = \mathbf{0}^t \iff \delta x_{i+\ell} = \mathbf{0}, \quad \forall i \in [r]. \tag{3.6}$$

Definition 3.6 (Compatible Vectors) *We call an m-dimensional vector $\overrightarrow{\delta}$, $\mathcal{U}$-*compatible *if*

$$\forall i \in [r],\ \delta_{i+\ell} = \mathbf{0} \text{ if and only if } \overrightarrow{z^{(i)}} = \mathbf{0}^t, \text{ where } \mathbf{u}^{(i)}(\delta_1, \ldots, \delta_m) = \overrightarrow{z^{(i)}}.$$

Moreover, it is called a ***collision-compatible vector*** *or* ***collision $\mathcal{U}$-compatible*** *if $\overrightarrow{\delta}$ is non-zero and $\delta_m = \mathbf{0}$. If $\overrightarrow{\delta}$ is non-zero $\mathcal{U}$-compatible, then $\delta_{[\ell]}$ is also non-zero (since, otherwise, the whole vector becomes zero vector using the definition of compatibility.*

Now, we show that the existence of a collision-compatible vector can lead to a collision of the hash mode provided the underlying oracles satisfy certain input-output constraints. More formally, we have the following lemma.

Lemma 3.2 *Let $\overrightarrow{\delta}$ be a $\mathcal{U}$-compatible m-dimensional vector for an (ℓ, r, t) linear hash mode $H := (\mathcal{U}, g)$. $\mathcal{O}_i(\overrightarrow{y^{(i)}}) = x_{i+\ell}$ and $\mathcal{O}_i(\overrightarrow{y'^{(i)}}) = x'_{i+\ell}$ where $\mathcal{U}(\overrightarrow{x}) = \mathcal{Y}$ and $\mathcal{U}(\overrightarrow{x'}) = \mathcal{Y}'$, we have*

$$\begin{aligned} \text{OUT}_H^{\mathcal{O}^r}(M) &= \overrightarrow{x} = (x_1, \ldots, x_{\ell+r}) \\ \text{OUT}_H^{\mathcal{O}^r}(M') &= \overrightarrow{x'} = (x'_1, \ldots, x'_{\ell+r}). \end{aligned}$$

Moreover, we have the following:

1. *If $\overrightarrow{\delta}$ is a collision-compatible, then*

$$\text{SLH}_{\mathcal{U}}^{\mathcal{O}_1, \ldots, \mathcal{O}_r}(M) = x_{\ell+r} = x'_{\ell+r} = \text{SLH}_{\mathcal{U}}^{\mathcal{O}_1, \ldots, \mathcal{O}_r}(M').$$

2. If $g(\overrightarrow{\delta}) = \mathbf{0}$*, then*

$$H^{\mathcal{O}_1,\ldots,\mathcal{O}_r}(M) = g(\overrightarrow{x}) = g(\overrightarrow{x'}) = H^{\mathcal{O}_1,\ldots,\mathcal{O}_r}(M'),.$$

The proof of the above lemma is straightforward from the definition. The last statement on the intermediate output is easy to observe. From this, the collision of hash output follows as the hash function is simple hash. We also note that in the above lemma, $M \neq M'$. Since otherwise, the intermediate inputs of M and M' are the same and hence contradicting $\overrightarrow{x} \oplus \overrightarrow{x'} = \overrightarrow{\delta} \neq \mathbf{0}^m$. This Lemma describes collision pairs of a linear hash mode $H = (\mathcal{U}, g)$, given that a collision-compatible differential vector $\overrightarrow{\delta}$ exists.

Now, we are ready to prove our main result with the above definitions and results. We know that the linear hash mode can be classified into two types based on its structure: simple and non-simple. Therefore, we first prove Theorem 3.2 for simple linear hash mode, followed by non-simple linear hash mode. More formally, for simple linear hash mode, we aim to prove the following statement:

Proposition 3.3 ***[Proposition for Simple Linear Hash Mode]*** *Let* $\textsc{slh}_{\mathcal{U}}$ *be an* (ℓ, r, t) *simple linear hash mode satisfying either (i)* $t \geqslant 2$*,* $r < \lceil(\ell-1)/(t-1)\rceil$*, or (ii)* $t = 1$*,* $\ell \geqslant 2$ *with* $(tr + \ell + r) < 2^n$*. Then, there exists*

- *t-to-1 collision-resistant compression functions* $f_1(1^n, \cdot), \ldots, f_r(1^n, \cdot)$ *and*
- *an uniform collision finder* A *such that* $\boldsymbol{CP}_{H,n}(A) = 1, \forall^* n$ *where* $H = \textsc{slh}_{\mathcal{U}}^{f_1,\ldots,f_r}$*.*

In other words, the white-box (and hence black-box) reduction from H *to* $(f_1, \ldots, f_r)$ *does not exist.*

Proof of Proposition 3.3. $\textsc{slh}_{\mathcal{U}}$ be an (ℓ, r, t)-linear simple hash mode satisfying either (i) $t \geqslant 2$, $r < \lceil(\ell-1)/(t-1)\rceil$, or (ii) $t = 1$, $\ell \geqslant 2$ with $(tr + \ell + r) < 2^n$. Therefore, by definition, we can represent $\textsc{slh}_{\mathcal{U}}$ as $\mathcal{U} = (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)})$, where $\mathcal{U}$ be a triangular dependent tuple of m-to-t linear function with $m = \ell + r$. Therefore, we state and prove the following Lemma:

Lemma 3.3 *Let* $\mathcal{U} := (\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(r)})$ *be a triangular dependent tuple of* m*-to-*t *linear function where* $m = r + \ell$ *and either (i)* $t \geqslant 2$*,* $r < \lceil(\ell-1)/(t-1)\rceil$*, or (ii)* $t = 1$*,* $\ell \geqslant 2$ *with* $(tr + \ell + r) < 2^n$*. Then, a collision* $\mathcal{U}$*-compatible vector exists, which can be computed efficiently.*

The proof of this lemma is intricate and lengthy, requiring several steps. Consequently, we have dedicated a separate section (Section 3.4) to its proof.

Now, we can see the above Lemma (Lemma 3.3) ensures the existence of efficiently computable collision $\mathcal{U}$-compatible vector $\overrightarrow{\delta}$. We split $\overrightarrow{\delta}$ as $\overrightarrow{x} \oplus \overrightarrow{x'} = \overrightarrow{\delta}$. Then, by using Lemma 3.2, we have an efficient algorithm A (which simply returns (M, M') computed from $\overrightarrow{x}, \overrightarrow{x'}$ respectively) such that $\mathbf{CP}_{H,n}(A) = 1$. This holds for any hash function as long as

oracles satisfy certain input-output constraints. By Proposition 3.1, we have such input-output fixing collision-resistant compression functions. Note that the vectors $\overrightarrow{x}, \overrightarrow{x'}$ can be computed efficiently (given the description of linear hash mode).

Now, it remains to prove Theorem 3.2 for the case of non-simple linear hash mode. Precisely, we prove the following:

Proposition 3.4 ***[Proposition for Non-simple Linear Hash Mode]*** *Let $H := \mathrm{CLH}_{(\mathcal{U},g)}$ be an (ℓ, r, t) non-simple linear hash mode satisfying either (i) $t \geqslant 2$, $r < \lceil (\ell-1)/(t-1) \rceil$, or (ii) $t = 1$, $\ell \geqslant 2$ with $(tr + \ell + r) < 2^n$. Then, there exists*

- *t-to-1 collision-resistant compression functions $f_1(1^n, \cdot), \ldots, f_r(1^n, \cdot)$ and*
- *an uniform collision finder A such that $\boldsymbol{CP}_{H,n}(A) = 1, \forall^* n$ where $H = \mathrm{CLH}^{f_1,\ldots,f_r}_{(\mathcal{U},g)}$.*

In other words, the white-box (and hence black-box) reduction from H to $(f_1, \ldots, f_r)$ does not exist.

Proof of Proposition 3.4. For non-simple linear hash mode $\mathrm{CLH}_{(\mathcal{U},g)}$ satisfying the conditions specified above, we can choose the index set $Z = \varnothing$, and construct the following two sets of vectors as follows:

$$\mathcal{S} = \{\overrightarrow{g}\}$$
$$\mathcal{N} = \{\overrightarrow{u_j^{(i)}} : i + \ell \in [\ell + r], j \in [t]\} \cup \{\overrightarrow{e_k} \ : \ k \in [\ell + r]\}$$

Note that $\mathsf{span}(\mathcal{S})$ and $\mathcal{N}$ are disjoint as $g(x_1, \ldots, x_{\ell+r})$ has non-zero coefficient for $x_{\ell+r}$, whereas $x_{\ell+r}$ is not present in any vector of $\mathcal{N}$. Now $\mathcal{N}$ contains $(tr + \ell + r) < 2^n$ vectors. Hence, using Lemma 3.1, there always exists $\overrightarrow{\delta} \in \mathsf{null}(\mathcal{S})$ such that $\forall \overrightarrow{v} \in \mathcal{N}$, $\overrightarrow{\delta} \cdot \overrightarrow{v} \neq \mathbf{0}$ (and hence for all $i \in [\ell + r]$, $\delta_i \neq \mathbf{0}$) and $g(\overrightarrow{\delta}) = \mathbf{0}$. We already know that the vectors $\overrightarrow{u_j^{(i)}}$ belongs to $\mathcal{N}$, for all $i + \ell \in [\ell + r]$, then we can easily conclude that, for all $j \in [t]$,

$$\overrightarrow{u_j^{(i)}} \cdot \overrightarrow{\delta} \neq \mathbf{0} \Rightarrow \mathbf{u}^{(i)}(\overrightarrow{\delta}) \neq \mathbf{0}^t \Rightarrow \overrightarrow{\beta^{(i)}} \neq \mathbf{0}^t$$

where $\mathcal{U}(\overrightarrow{\delta}) = (\overrightarrow{\beta^{(1)}}, \ldots, \overrightarrow{\beta^{(r)}})$. This immediately implies that $\overrightarrow{\delta}$ is $\mathcal{U}$-compatible (since all intermediate input and output vectors are non-zero).

Now we split $\overrightarrow{\delta} = \overrightarrow{x} \oplus \overrightarrow{x'}$. Note that the vectors $\overrightarrow{x}, \overrightarrow{x'}$ can be computed efficiently (given the description of linear hash mode). Therefore, by using Lemma 3.2 and Proposition 3.1 (similar to the proof of the simple linear has mode), we can have an efficient algorithm A, which returns a message pair (M, M') computed from $\overrightarrow{x}$, $\overrightarrow{x'}$ such that $\mathbf{CP}_{H,n}(A) = 1$. Hence, we conclude the proof of Theorem 3.2.

Non-Uniform Setting. Based on the collision-resistant assumption against a non-uniform collision finder, we can state our result as follows:

Theorem 3.5 *Let H be an (ℓ, r, t) linear hash mode satisfying either (i) $t \geqslant 2$, $r < \lceil(\ell - 1)/(t-1)\rceil$, or (ii) $t = 1$, $\ell \geqslant 2$ with $(tr + \ell + r) < 2^n$ and let $\alpha := (\alpha_n)_n$ be an advise string. Then, there exists*

- *t-to-1 collision-resistant compression functions $f_1(1^n, \cdot), \ldots, f_r(1^n, \cdot)$ against α and*
- *an uniform collision find,er A such that $\boldsymbol{CP}_{H^{f_1,f_2,\ldots,f_r},n}(A) = 1, \forall^* n$.*

In other words, the white-box (and hence black-box) reduction from H to $(f_1, \ldots, f_r)$ does not exist.

Proof Overview. The proof approach is similar to the one we used in the uniform setup. First, we show that for an (ℓ, r, t)-linear hash mode satisfying the above conditions, a collision-compatible vector exists, which can be computed efficiently. Following this, we establish that the existence of a collision-compatible vector can lead to a collision of the hash mode provided the underlying oracles satisfy certain input-output constraints. Finally, to conclude the proof, we need to construct some input-output fixing collision-resistant compression functions $f_1, f_2, \ldots, f_r$ against advise string α, such that $H^{f_1,f_2,\ldots,f_r}$ is not collision-resistant. Hence, we prove the following proposition:

Proposition 3.6 *Let (μ, ν) be a (t, r) in-out computable pair. There is a $\mu \mapsto \nu$ input-output fixing collision-resistant compression function against an advise string α provided the collision-resistant assumption against non-unifrom collision finder is true.*

The proof idea is similar to the proof of Proposition 3.1. Let h' be a c-to-1 collision-resistant compression function against an advise string α (it exists by our collision-resistant assumption against non-uniform collision finder), where $c \geqslant 2t$ be some fixed integer. Then, we define a $\mu \mapsto \nu$ input-output fixing mapping $h_{\mu,\nu}$ similarly as in Equation 3.1. Finally, we prove $h_{\mu,\nu}$ is collision-resistant, provided h' is collision-resistant.
Thus, we arrive at the conclusion of the proof of Theorem 3.5.

3.4 Proof of Lemma 3.3

$\mathcal{U}$-consistent Index set. Let $Z = \{i \in [\ell + r] : x_i = \mathbf{0}\}$ be called a *zero-index* set of $\overrightarrow{x}$. Now, for all $i + \ell \in Z$, $\mathbf{u}^{(i)}(\overrightarrow{x}) = \mathbf{0}^t$ and so for all $1 \leqslant j \leqslant t$, $\overrightarrow{u_j^{(i)}}|_Z \cdot \overrightarrow{x} = \mathbf{0}$. Let

$$\mathcal{S} = \{\overrightarrow{u_j^{(i)}}|_Z : i + \ell \in Z, j \in [t]\}.$$

Thus, if for some $k \in [\ell + r]$, $\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S})$ then $x_k = \overrightarrow{e_k} \cdot \overrightarrow{x} = \mathbf{0}$, and hence $k \in Z$. So, we state our first necessary condition:

N1: $\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}) \Rightarrow k \in Z$.

By using the similar argument (using the condition, $\overrightarrow{y^{(k)}} = \mathbf{0}^t$ if and only if $k + \ell \in Z$) we have

N2: $i+\ell \notin Z \Rightarrow \exists j \in [t], \overrightarrow{u_j^{(i)}}|_Z \notin \mathsf{span}(\mathcal{S})$. A contra-positive statement says that $\forall j \in [t], \overrightarrow{u_j^{(i)}}|_Z \in \mathsf{span}(\mathcal{S}) \Rightarrow i+\ell \in Z$.

Any set Z satisfying the above two necessary conditions is called $\mathcal{U}$-consistent or simply *consistent* whenever the tuple $\mathcal{U}$ is understood. Moreover, Z is called *non-trivial* if $Z \neq [\ell+r]$ (note that $[\ell+r]$ is trivially consistent which leads to the compatible vector $\mathbf{0}^{\ell+r}$). Now, we show that the above necessary conditions are indeed sufficient.

Lemma 3.4 (Linear Algebra Technical Lemma) *Let $\mathcal{U}$ be a triangular dependent tuple of $\ell+r$-to-t linear functions such that $tr+\ell+r<2^n$. If $Z \subseteq [\ell+r]$ is a non-trivial $\mathcal{U}$-consistent index set, then there exists a non-zero $\mathcal{U}$-compatible vector $\overrightarrow{\alpha}$ with the zero-index set as Z.*

Proof. We set $Z^c = [\ell+r]\backslash Z$. We prove this result using Lemma 3.1. To do so, we define:

$$I = \{(i+\ell, j) \in Z^{\mathsf{c}} \times [t] \ : \ \overrightarrow{u_j^{(i)}}|_Z \notin \mathsf{span}(\mathcal{S})\}$$
$$\mathcal{S} = \{\overrightarrow{u_j^{(i)}}|_Z : i+\ell \in Z, j \in [t]\}$$
$$\mathcal{S}' = \{\overrightarrow{u_j^{(i)}}|_Z : (i+\ell, j) \in I\}$$

We moreover define $\mathcal{S}_1 = \mathcal{S} \cup \{\overrightarrow{e_k} \ : \ k \in Z\}$ and $\mathcal{N} = \mathcal{S}' \cup \{\overrightarrow{e_k} \ : \ k \notin Z\}$.

Claim. $\mathsf{span}(\mathcal{S}_1)$, and $\mathcal{N}$ are disjoint.

Proof of Claim. We prove this by contradiction. Let there exist some $\overrightarrow{v} \in \mathcal{N}$ such that $\overrightarrow{v} \in \mathsf{span}(\mathcal{S}_1)$. This implies *either* $\overrightarrow{v} = \overrightarrow{u_j^{(i)}}|_Z$, for some $(i+\ell, j) \in I$ *or* $\overrightarrow{v} = \overrightarrow{e_k}$, for some $k \notin Z$. Now, if $\overrightarrow{v} = \overrightarrow{u_j^{(i)}}|_Z$, for some $(i+\ell, j) \in I$, then $\overrightarrow{v} = \overrightarrow{u_j^{(i)}}|_Z \in \mathsf{span}(\mathcal{S}_1)$. Therefore, we can write

$$\overrightarrow{u_j^{(i)}}|_Z = \sum_{(p+\ell,q) \in Z \times [t]} c_{p,q} \overrightarrow{u_q^{(p)}}|_Z \ + \sum_{s \in Z} c_s \overrightarrow{e_s}$$

where $c_{p,q}, c_s \in \mathbb{F}$. Now, if $c_s = \mathbf{0}$, $\forall s \in Z$, then $\overrightarrow{u_j^{(i)}} \in \mathsf{span}(\mathcal{S})$ which cannot be true as $(i+\ell, j) \in I$. Furthermore, if there exists some $s \in Z$ s.t. $c_s \neq \mathbf{0}$, then $\overrightarrow{v}$ contains some non-zero entry in the s-th index which cannot be possible. So,

$$\overrightarrow{v} = \overrightarrow{u_j^{(i)}}|_Z \notin \mathsf{span}(\mathcal{S}_1), \text{ for all } (i+\ell, j) \in I.$$

On the other hand, if $\overrightarrow{v} = \overrightarrow{e_k}$, for some $k \notin Z$, then $\overrightarrow{v} = \overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S})$ using the condition N1. But according to our assumption $\overrightarrow{v} = \overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1)$, where $k \notin Z$. Therefore, we write,

$$\overrightarrow{e_k} = \sum_{(i+\ell,j) \in Z \times [t]} d_{i,j} \overrightarrow{u_j^{(i)}}|_Z \ + \sum_{q \in Z} d_q \overrightarrow{e_q}$$

where $d_{i,j}, d_q \in \mathbb{F}$. As, $\overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S})$, then there exists some $q \in Z$ s.t $d_q \neq \mathbf{0}$. However, this cannot be true, as the vector $\overrightarrow{e_k}$ does not have any non-zero entry at the q-th index where $q \in Z$. So,

$$\overrightarrow{v} = \overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S}_1), \text{ for all } k \notin Z.$$

This completes the proof of the claim.

Note, $\mathcal{N}$ can have at most $(tr + \ell + r) < 2^n$ vectors. Hence, using Lemma 3.1, there exists $\overrightarrow{\alpha} \in \mathsf{null}(\mathcal{S}_1)$ such that $\forall \overrightarrow{v} \in \mathcal{N}$, $\overrightarrow{v} \cdot \overrightarrow{\alpha} \neq \mathbf{0}$. Now, as $\overrightarrow{\alpha} \in \mathsf{null}(\mathcal{S}_1)$, then we have, for all $k \in Z$,

$$\overrightarrow{e_k} \cdot \overrightarrow{\alpha} = \mathbf{0} \Rightarrow \alpha_k = \mathbf{0}.$$

Furthermore, as for all $\overrightarrow{v} \in \mathcal{N}$, $\overrightarrow{v} \cdot \overrightarrow{\alpha} \neq \mathbf{0}$, we have, for all $k \notin Z$,

$$\overrightarrow{e_k} \cdot \alpha \neq \mathbf{0} \Rightarrow \alpha_k \neq \mathbf{0}.$$

Hence, Z is zero-index set of $\overrightarrow{\alpha}$. As, $Z \neq [\ell + r]$, then $\overrightarrow{\alpha} \neq \mathbf{0}^m$. Now, the only remaining task is to prove that $\overrightarrow{\alpha}$ is $\mathcal{U}$-compatible. As, $\overrightarrow{\alpha} \in \mathsf{null}(\mathcal{S}_1)$, then we have, for all $(i + \ell) \in Z$ with $i \in [r]$,

$$\begin{aligned} & \overrightarrow{u_j^{(i)}}|_Z \cdot \overrightarrow{\alpha} = \mathbf{0}, \ \forall j \in [t] \\ \Rightarrow\ & \mathbf{u}^{(i)}(\overrightarrow{\alpha}) = \mathbf{0}^t \\ \Rightarrow\ & \overrightarrow{\beta^{(i)}} = \mathbf{0}^t \ \ [\text{Here we denote, } \mathcal{U}(\overrightarrow{\alpha}) = (\overrightarrow{\beta^{(1)}}, \ldots, \overrightarrow{\beta^{(r)}})]. \end{aligned}$$

Therefore, we have, for all $i \in [r]$, if $\alpha_{i+\ell} = \mathbf{0}$, then $\overrightarrow{\beta^{(i)}} = \mathbf{0}^t$. Finally, as we have for all $\overrightarrow{v} \in \mathcal{S}_2$, $\overrightarrow{v} \cdot \overrightarrow{\alpha} \neq \mathbf{0}$, then for all $(i + \ell) \notin Z$, there exists some $j \in [t]$, such that

$$\overrightarrow{u_j^{(i)}}|_Z \cdot \overrightarrow{\alpha} \neq \mathbf{0} \Rightarrow \mathbf{u}^{(i)}(\overrightarrow{\alpha}) \neq \mathbf{0}^t \Rightarrow \overrightarrow{\beta^{(i)}} \neq \mathbf{0}^t.$$

Thus, we can easily conclude that for all $i \in [r]$ with $(i + \ell) \notin Z$ i.e., $\alpha_{i+\ell} \neq \mathbf{0}$ implies $\overrightarrow{\beta^{(i)}} \neq \mathbf{0}^t$. This immediately implies that $\overrightarrow{\alpha}$ is $\mathcal{U}$-compatible. Hence, the result follows. □

Now, given such $\mathcal{U}$, we describe an efficient algorithm (Algorithm 2) that yields a non-trivial $\mathcal{U}$-consistent index set Z.

Algorithm 2 Calculating an index set Z from $\mathcal{U}$

Initialization: $Z = \{(\ell + r)\}$, $N = [\ell + r - 1]$, $\mathcal{S} = \{\overrightarrow{u_j^{(r)}}|_Z \ : \ j \in [t]\}$
while $\exists\ k \in N$ s.t $\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S})$ **or** $\overrightarrow{u_j^{(k-\ell)}}|_Z \in \mathsf{span}(\mathcal{S})$, $\forall j \in [t]$, and $k \geqslant (\ell + 1)$ **do**
    Choose the largest such k
    $Z \leftarrow Z \cup \{k\}$, $N \leftarrow N \backslash \{k\}$
    if $k = i + \ell$, for some $i \in [r]$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{\overrightarrow{u_j^{(i)}}|_Z \ : \ j \in [t]\}$
return Z

Algorithm 2. For a triangular dependent tuple of $\ell + r$-to-t linear functions $\mathcal{U}$, Algorithm 2 starts with two sets of indices Z and N such that Z contains only one index $(\ell + r)$, and N contains all the other indices in the set $[\ell + r]$. The set of vectors $\mathcal{S}$ contains the vectors $\overrightarrow{u_j^{(r)}}|_Z$, for all $j \in [t]$. Now, we continue to transfer the indices from N to Z if any index k in N satisfies any of the following two conditions:

$$\begin{cases} \textbf{Condition 1 :} \ \overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}). \\ \textbf{Condition 2 :} \ \overrightarrow{u_j^{(k-\ell)}}|_Z \in \mathsf{span}(\mathcal{S}), \ \forall j \in [t], \ k \geqslant (\ell + 1). \end{cases}$$

We can define a index set $Z_\ell \subseteq Z$ as follows:

$$Z_\ell = \{i + \ell \in Z \ : \ i \in [r]\}.$$

Therefore at any instance of Algorithm 2, the set of vectors $\mathcal{S} = \{\overrightarrow{u_j^{(i)}}|_Z \ : \ i + \ell \in Z_\ell, \ j \in [t]\}$. Furthermore, whenever we find that any index $k \in N$ satisfies at least one of the conditions above, we transfer k from N to Z. So, we can partition the index set Z_ℓ into two subsets as follows:

$$Z_\ell = Z_\ell^1 \sqcup Z_\ell^2.$$

Here, the index set Z_ℓ^i consisting of those indices k that are transferred from N to Z_ℓ after satisfying **Condition** $\boldsymbol{i}$, for all $i \in [2]$. Using the index set Z_ℓ^1, we define $\mathcal{S}_1$ as follows:

$$\mathcal{S}_1 = \{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell^1, \ j \in [t]\}$$

Next, we examine key properties derived from Algorithm 2 to ultimately prove the Lemma 3.3. The properties are as follows:

Property 3.1 $\mathsf{rank}(\mathcal{S}_1) \leqslant t|Z_\ell^1|$.

This property directly follows from the fact that, $\mathcal{S}_1$ contains total $t|Z_\ell^1|$ vectors which implies $\mathsf{rank}(\mathcal{S}_1)$ can be at most $t|Z_\ell^1|$.

Property 3.2 *For any index $k \in Z$, where $k \leqslant \ell$, we have,*

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}).$$

Property 3.3 *For any index $k \in Z \backslash \{\ell + r\}$, where $k \geqslant (\ell + 1)$, we have,*

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}).$$

Property 3.4 *Algorithm 2 always returns $\mathcal{U}$-consistent index set Z.*

The proof of these aforementioned properties has been deferred to the Subsection 3.4.1, respectively.

Finally, to establish the non-trivial $\mathcal{U}$-consistent nature of the index set Z constructed through Algorithm 2, we formulate and subsequently prove the following lemma:

Lemma 3.5 *Let $\mathcal{U}$ be a triangular dependent tuple of $\ell + r$-to-t linear function such that either (1) $r < \lceil (\ell-1)/(t-1) \rceil$, $t \geqslant 1$ or (2) $t = 1, \ell \geqslant 2$. If Z is an index set constructed using Algorithm 2, then Z is a non-trivial $\mathcal{U}$-consistent index set.*

Proof. Let $\mathcal{U}$ be a triangular dependent tuple of $\ell+r$-to-t linear function, and Z is the output of Algorithm 2. From Property 3.4, one can easily conclude that Z is $\mathcal{U}$-consistent index set. Now, we closely look into the Algorithm 2, Property 3.1, Property 3.2, and Property 3.3, and try to see some relations between the cardinalities of the index sets Z, and Z_ℓ.
We already know that the set of indices Z_ℓ^1 consisting of those indices k which are transferred from N to Z after satisfying **Condition 1**, and $\mathcal{S}_1$ is the set of vectors $\overrightarrow{u_j^{(i)}}$, for all $i + \ell \in Z_\ell^1$, and for all $j \in [t]$. Then, from Property 3.1, we have,

$$\mathsf{rank}(\mathcal{S}_1) \leqslant t|Z_\ell^1| \tag{3.7}$$

Moreover, from the properties of Algorithm 2, we can easily conclude that $\mathsf{span}\{\overrightarrow{e_i} \ : \ i \in Z \backslash \{\ell + r\}\}$ is a subspace of $\mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\})$. Therefore, taking the dimensions of both these vector spaces, we can easily conclude that

$$|Z| - 1 \leqslant \mathsf{rank}(\mathcal{S}_1) + |Z_\ell^2| \leqslant \mathsf{rank}(\mathcal{S}_1) + t|Z_\ell^2| \leqslant t|Z_\ell|$$

which leads us to the fact that $|Z| \leqslant t|Z_\ell| + 1$.
Now, we construct the proof using a contradiction-based method. Therefore, we assume that Z is a trivial $\mathcal{U}$-consistent index set i.e., $Z = [\ell + r]$. Now, the following two cases arise:

Case-A If $t = 1$, and $Z = [\ell + r]$, then we can easily say that $|Z| \leqslant |Z_\ell| + 1$, which implies

$$\ell + r \leqslant r + 1 \Rightarrow \ell \leqslant 1$$

However, it contradicts the given condition, as we know that if $t = 1$, then $\ell \geqslant 2$.

Case-B In this case, as $t \geqslant 2$, and $Z = \ell + r$ which directly implies that

$$|Z| \leqslant t|Z_\ell| + 1 \Rightarrow \ell + r \leqslant tr + 1 \Rightarrow r \geqslant (\ell - 1)/(t - 1)$$

It is given that when $t > 1$, it should be $r < \lceil (\ell-1)/(t-1) \rceil$,, and we arrive at a contradiction. Therefore, Z is non-trivial $\mathcal{U}$-consistent index set. □

Now, from Algorithm 2, and Lemma 3.5, we have that for a triangular dependent tuple $\mathcal{U}$ of $\ell + r$-to-t linear function, satisfying either (1) $r < \lceil (\ell - 1)/(t - 1) \rceil$, $t > 1$ or (2) $t = 1, \ell \geqslant 2$ with $(tr + \ell + r) < 2^n$, one can construct a non-trivial $\mathcal{U}$-consistent index set Z with at least one index $(\ell + r)$. Therefore, using Lemma 3.4 (*Linear Algebra Technical Lemma*), we can easily conclude that there exists a non-zero $\mathcal{U}$-compatible vector $\overrightarrow{\alpha}$ with the zero index set as Z.
Furthermore, as $\overrightarrow{\alpha} \neq \mathbf{0}^{\ell+r}$ is $\mathcal{U}$-compatible, then $\overrightarrow{\alpha}_{[\ell]} \neq \mathbf{0}^\ell$. Furthermore, as the index set Z contains $m = \ell + r$, and Z is the zero-index set of $\overrightarrow{\alpha}$, therefore, $\overrightarrow{\alpha}$ is non-zero collision $\mathcal{U}$-compatible vector. Hence, the result follows.

3.4.1 Proof of Properties

In this section, we prove the properties stated above which have been used to prove our Proposition.

Proof of Property 3.2

Consider that instance of Algorithm 2, when the index set Z contains $k \geqslant \ell+1$ which implies that $Z = Z_\ell$. At that instance, suppose for the first time one find out some index $k \leqslant \ell$ such that,

$$\begin{aligned} & \overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}) \\ \Rightarrow\ & \overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}}|_Z \ : \ i \in Z_\ell,\ j \in [t]\}) \\ \Rightarrow\ & \overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell,\ j \in [t]\} \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell\}) \end{aligned}$$

Now, if $Z = Z_\ell = \mathcal{Z}_\ell^1$, then we know that for all $i \in Z_\ell^1$, $\overrightarrow{e_i} \in \mathsf{span}(\mathcal{S}_1)$. Therefore, we have,

$$\overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell^1,\ j \in [t]\}) = \mathsf{span}(\mathcal{S}_1).$$

Next, if $Z = Z_\ell = Z_\ell^1 \cup Z_\ell^2$, then we have,

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\})$$

as we know if $Z = Z_\ell$, then for all $i+\ell \in Z_\ell^2$, and $j \in [t]$, we have $\overrightarrow{u_j^{(i)}} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\})$ (Lemma 3.6). Therefore, according to Algorithm 2, we transfer the index k from N to Z, and update $\mathcal{S}$ accordingly.

From here, we can easily conclude that, for any index $k \in Z$, where $k < \ell + 1$, then,

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}).$$

Hence, the result follows.

Proof of Property 3.3

At any instance of Algorithm 2, the index set Z can have indices $i \geqslant (\ell + 1)$ which are transferred from N to Z after satisfying both the conditions, and some indices k satisfying $k \leqslant \ell$ as well.[9] Therefore, for any index $k \in Z_\ell^1 \backslash \{(\ell + r)\}$, we have,

$$\begin{aligned} & \overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell\} \cup \{\overrightarrow{e_i} \ : \ i \in Z \backslash \{k\}\}) \\ \Rightarrow\ & \overrightarrow{e_k} \in \mathsf{span}(\{\overrightarrow{u_j^{(i)}} \ : \ i \in Z_\ell^1\} \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}) \text{ [Using Property 3.2]} \\ \Rightarrow\ & \overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}) \end{aligned}$$

Hence, the result follows.

[9] At any instance of Algorithm 2, any index $k \leqslant \ell$ can be transferred from N to Z after satisfying **Condition 1** only.

Proof of Property 3.4

Let Algorithm 2 return an index set say Z. Therefore, from Algorithm 2, we have, for all $k \notin Z$, $\overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S})$, and for all $k \notin Z$ where $k \geqslant (\ell+1)$, there exists $j \in [t]$ such that $\overrightarrow{u_j^{(k-\ell)}}|_Z \notin \mathsf{span}(\mathcal{S}$. Now, for all $k \notin Z$, $\overrightarrow{e_k} \notin \mathsf{span}(\mathcal{S})$ implies that the index set satisfies the first necessary condition N1:

$$\overrightarrow{e_k} \in \mathsf{span}(\mathcal{S}) \Rightarrow k \in Z.$$

Furthermore, for all $i+\ell \notin Z$, there exists $j \in [t]$ such that $\overrightarrow{u_j^{(k-\ell)}}|_Z \notin \mathsf{span}(\mathcal{S})$ immediately implies that the index set satisfies the second necessary condition N2. Hence, Z is $\mathcal{U}$-consistent index set.

Proof of Lemma 3.6

Lemma 3.6 *If the index set $Z = Z_\ell$, then for all $k+\ell \in Z_\ell^2$, and $j \in [t]$, we have,*

$$\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}).$$

Proof. Consider that instance of Algorithm 2, when the index set $Z = Z_\ell^1$ which means at that instance Z_ℓ^2 is empty. Now, at that instance, suppose for the first time we find out some index $k_1 = k+\ell \geqslant \ell+1$ such that,

$$\begin{aligned}
&\overrightarrow{u_j^{(k)}}|_Z \in \mathsf{span}(\mathcal{S}), \ \forall j \in [t] \\
\Rightarrow\ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\{\overrightarrow{u_{j_1}^{(i)}} \ : \ i \in Z_\ell^1, \ j_1 \in [t]\} \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^1\}), \ \forall j \in [t] \\
\Rightarrow\ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\{\overrightarrow{u_{j_1}^{(i)}} \ : \ i \in Z_\ell^1, \ j_1 \in [t]\}), \ \forall j \in [t] \\
\Rightarrow\ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\mathcal{S}^1), \ \forall j \in [t]
\end{aligned}$$

Therefore, according to Algorithm 2 we transfer the index k_1 from N to Z (precisely the index k_1 is added to Z_ℓ^2), and update $\mathcal{S}$ accordingly.

Next, if we consider any instance of Algorithm 2, where the index set $Z = Z_\ell$, then for all $k_1 = k+\ell \in Z_\ell^2$,

$$\begin{aligned}
\Rightarrow\ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\{\overrightarrow{u_{j_1}^{(i)}} \ : \ i \in Z_\ell^1, \ j_1 \in [t]\} \cup \{\overrightarrow{e_i} \ : \ i \in \mathcal{Z}_\ell\}), \ \forall j \in [t] \\
\Rightarrow\ &\overrightarrow{u_j^{(k)}} \in \mathsf{span}(\mathcal{S}_1 \cup \{\overrightarrow{e_i} \ : \ i \in Z_\ell^2\}), \ \forall j \in [t]
\end{aligned}$$

Hence, the result follows. □

3.5 Collision-Resistance Preserving Security Bound for Linear Hash Mode having s ($s \geqslant 2$) Output Blocks

In this section, we extend our lower bound result for hash functions with multiple hash outputs. The main idea of the lower bound is to convert a s-output hash function to a single

block hash function by applying a suitable postprocessor. Then we can apply our previous result to obtain a lower bound on the compression function calls of s-output hash function for collision-resistance preserving property. Precisely, we state and prove the following theorem:

Theorem 3.7 *Let $H : \mathbb{F}^\ell \to \mathbb{F}^s$ be a linear hash mode which computes $H(M)$ for a given message $M = (m_1, m_2, \ldots, m_\ell) \in \mathbb{F}^\ell$ via access to r many t-to-1 compression function calls with $t, s \geqslant 2$, and $(tr + \ell + r) < 2^n$. If*

$$r < \lceil (\ell - s)/(t-1) \rceil,$$

then this construction does not have white-box (and hence black-box) reduction.

Proof. We prove this result by contradiction. Let, $H : \mathbb{F}^\ell \to \mathbb{F}^s$ be a linear hash mode with $r < \lceil (\ell - s)/(t-1) \rceil$, t-to-1 compression function calls and H has white-box (black-box) reduction.
As we know that given any integers a and b, with $a > 0$, there exist unique integers p and q such that $b = pa + q$, $0 \leqslant q < a$. So, for the integers $(\ell - 1), (s - 1)$, and $(t - 1)$, where $t \geqslant 2$, there exist unique integers p, p', q, and q' such that

$$\begin{cases} \ell - 1 = (t-1)p + q \\ s - 1 = (t-1)p' + q' \\ \ell - s = (t-1)(p - p') + (q - q') \end{cases} \tag{3.8}$$

where $0 \leqslant q, q' < (t-1)$. Therefore, from Equation 3.8, we can easily conclude the following:

$$\begin{cases} \lceil (\ell - s)/(t-1) \rceil = \lceil (\ell - 1)/(t-1) \rceil - \lceil (s-1)/(t-1) \rceil & \text{if either } q = q' \text{ or } q' > q > 0 \\ & \text{or } q > 0,\ q' = 0 \\ \lceil (\ell - s)/(t-1) \rceil > \lceil (\ell - 1)/(t-1) \rceil - \lceil (s-1)/(t-1) \rceil & \text{if either } q = 0,\ q' > 0 \\ & \text{or } q > q' > 0 \end{cases} \tag{3.9}$$

We will now examine these two cases in detail. Suppose,

$$\lceil (\ell - s)/(t-1) \rceil = \lceil (\ell - 1)/(t-1) \rceil - \lceil (s-1)/(t-1) \rceil. \tag{3.10}$$

Therefore from H, we can construct $H_1 : \mathbb{F}^\ell \to \mathbb{F}$ using $r_1 = r + \lceil (s-1)/(t-1) \rceil$ many t-to-1 compression function calls, same as in (Figure 3.7):

$$H_1(m_1, m_2, \ldots, m_\ell) = MD(H(m_1, m_2, \ldots, m_\ell)) \tag{3.11}$$

where MD is the Merkle-Damgård hash structure, which takes s block messages and outputs a message of length n using $\lceil (s-1)/(t-1) \rceil$ many t-to-1 compression function calls.
As, we know that both H and the Merkle-Damgård hash functions have a black-box (hence white-box) reduction (as existence of black-box reduction implies the existence of white-box reduction), then we can easily prove that H_1 defined in Equation 3.11 has black-box (hence white-box) reduction using r_1 many t-to-1 compression function calls. As, $r < \lceil (\ell - s)/(t-1) \rceil$, therefore,

$$r_1 < \lceil (\ell - s)/(t-1) \rceil + \lceil (s-1)/(t-1) \rceil$$

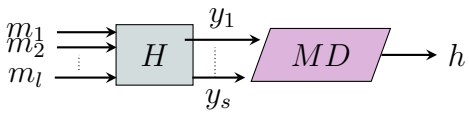


Figure 3.7: Linear Hash Mode H_1 using H.

which implies $r_1 < \lceil(\ell-1)/(t-1)\rceil$ (using Equation 3.10).

If $\lceil(\ell-s)/(t-1)\rceil > \lceil(\ell-1)/(t-1)\rceil - \lceil(s-1)/(t-1)\rceil$, in this case $q' > 0$ (Equation 3.9) which means $(s-1)$ is not multiple of $(t-1)$. For this case, we can have a integer k such that $q'+k = t-1$. Therefore, we can write $s+k-1 = (t-1)(p'+1)$. Now, all together we have

$$\begin{cases} s+k-1 = (t-1)(p'+1) \\ \ell - s = (t-1)(p-p') + (q-q') \\ \ell + k - 1 = (t-1)(p+1) + (q-q') \end{cases} \tag{3.12}$$

Hence, we can easily conclude that,

$$\lceil(\ell-s)/(t-1)\rceil = \lceil(\ell+k-1)/(t-1)\rceil - \lceil(s+k-1)/(t-1)\rceil.$$

Then, from H, we will construct $H_2 : \mathbb{F}^{(\ell+k)} \to \mathbb{F}$ using r_2 many t-to-1 compression function calls in the following way:

$$H_2(m_1, m_2, \ldots, m_{\ell+k}) = MD(H(m_1, m_2, \ldots, m_\ell), m_{\ell+1}, \ldots, m_{\ell+k}) \tag{3.13}$$

where MD is the Merkle-Damgård hash function which takes here $s+k$ block messages and uses $\lceil(s+k-1)/(t-1)\rceil$ compression function calls (Figure 3.8), i.e.,

$$r_2 = r + \lceil(s+k-1)/(t-1)\rceil.$$

Therefore, as we know that both H and the Merkle-Damgård hash have black-box (hence white-box) reduction, we can easily prove that H_2 defined in Equation 3.13 has black-box (hence white-box) reduction using r_2 many t-to-1 compression function calls. As, $r < \lceil(\ell-s)/(t-1)\rceil$, therefore we can conclude that

$$r_2 < \lceil(\ell-s)/(t-1)\rceil + \lceil(s+k-1)/(t-1)\rceil = \lceil(\ell+k-1)/(t-1)\rceil$$

which implies that from H we can construct a linear hash mode $H_2 : \mathbb{F}^{(\ell+k)} \to \mathbb{F}$ which has black-box (hence white-box) reduction using r_2 many t-to-1 compression function call where $r_2 < \lceil(\ell+k-1)/(t-1)\rceil$.

Thus, in both the scenarios, we have demonstrated that if there exists a linear hash mode, denoted as $H : \mathbb{F}^\ell \to \mathbb{F}^s$, with r t-to-1 compression function calls, where $r < \lceil(\ell-s)/(t-1)\rceil$, and H possesses black-box (hence white-box) reduction, then it is possible to construct another linear hash mode, denoted as $H' : \mathbb{F}^{\ell'} \to \mathbb{F}$, with black-box reduction (hence white-box reduction) using r' t-to-1 compression function calls, where $r' < \lceil(\ell'-1)/(t-1)\rceil$, for some $\ell' > \ell$. However, this construction contradicts the result we proved earlier in Theorem 3.2. Hence, the result follows. □

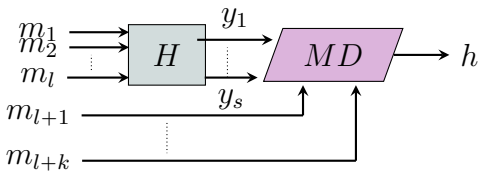


Figure 3.8: Linear Hash Mode H_2 using $r + \lceil (s+k-1)/(t-1) \rceil$ many t-to-1 Compression Function Calls where MD is Merkle-Damgård hash function.

3.5.1 Tightness of Lower Bound

To construct a linear hash mode $H : \mathbb{F}^\ell \to \mathbb{F}^s$, we design it in such a way that processing an ℓn-bit input necessitates $\lceil (\ell - s)/(t-1) \rceil$ calls to t-to-1 compression functions. Subsequently, we establish the proof that H possesses black-box reduction, where ℓ, s, and n are integers greater than or equal to 1. We construct $H : \mathbb{F}^\ell \to \mathbb{F}^s$ in the following way (Figure 3.9):

$$H(m_1, m_2, \ldots, m_\ell) = (MD(m_1, m_2, \ldots, m_{\ell-s+1}), m_{\ell-s+2}, \ldots, m_\ell) \tag{3.14}$$

where MD is the Merkle-Damgård hash function which takes here $\ell - s + 1$ block messages and uses $\lceil (\ell-s)/(t-1) \rceil$ many t-to-1 compression function calls. So, we prove that H (defined in Equation 3.14) has black-box reduction as follows:

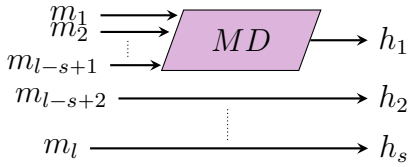


Figure 3.9: Linear Hash Mode H using $\lceil (l-s)/(t-1) \rceil$ many t-to-1 Compression Function Calls.

Lemma 3.7 *Given that the Merkle-Damgård hash function has a black-box reduction, H defined in Equation 3.14 also has a black-box reduction.*

Proof. Assuming H defined in Equation 3.14 does not have black-box collision reduction whenever $(M, M') \in \mathbb{F}^\ell \times \mathbb{F}^\ell$ is a collision pair of $H^{f_1, f_2, \ldots, f_r}$ for some collision-resistant t-to-1 compression functions $f_1, f_2, \ldots, f_r$ where $r = \lceil (\ell - s)/(t-1) \rceil$, $(M_1, M_2) = (m_1, m_2, \ldots, m_{\ell-s+1}, m'_1, m'_2, \ldots, m'_{\ell-s+1}) \in \mathbb{F}^{(\ell-s+1)} \times \mathbb{F}^{(\ell-s+1)}$ is a collision pair of Merkle-Damgård hash mode processing $\ell - s + 1$ block messages using $r = \lceil (\ell-s)/(t-1) \rceil$ calls to the underlying compression function. This leads us to the conclusion that the Merkle-Damgård hash mode does not have a black-box reduction, which is a contradiction. Hence the result follows. □

Therefore, using the result proved in Lemma 3.7, and from the construction defined in Equation 3.14, we have the result that for a linear hash mode $H : \mathbb{F}^\ell \to \mathbb{F}^s$ to have black-box reduction, it should call at least $\lceil (\ell - s)/(t-1) \rceil$ many t-to-1 compression functions. Please note that, the existence of black-box reduction implies the existence of white-box reduction.

3.6 Conclusion

In this chapter, we have initially demonstrated the impossibility of proving collision security for the recently constructed hash modes ABR [9] and T_5 [8] solely based on the collision security assumption of the underlying compression function. Despite T_5 being proven collision-resistant under the random oracle model, we have established a lower bound on the number of calls to the underlying compression function required to achieve collision-resistance preserving properties of a hash function when assuming the intermediate processing functions possess linear characteristics.

Chapter 4

Towards Finding Complete Impossible Differential Attacks on AndRX and ARX Designs

Impossible differential (ID) cryptanalysis, recognized as one of the most potent methods for breaking block ciphers, was separately pioneered by Biham *et al.* [82] and Knudsen [83]. The central concept involves using impossible differentials—propagations with zero probability—to differentiate the block cipher from a random permutation. Once an ID distinguisher is established, it is extended by a few rounds. Any key guesses that result in a pair matching the impossible differential are considered incorrect and discarded. With access to a specific number of pairs, the objective is to eliminate as many incorrect key candidates as possible during the *guess-and-filter* step. The remaining candidates are then brute-forced in the *exhaustive search step* to identify the correct key. By adjusting parameters such as the number of pairs, one can balance the complexity between the guess-and-filter and exhaustive search phases. The ID attack has been effectively utilized against numerous block ciphers. For instance, it provides the best cryptanalysis results for CAMELLIA [44], [84]. Moreover, the ID attack was the pioneering method to break seven rounds of the AES cipher [85]–[87], and it remained one of the most effective attacks for a considerable duration. In the realm of linear cryptanalysis, the dual to the ID attack is the zero-correlation (ZC) attack, introduced by Bogdanov and Rijmen [49]. Whereas the ID attack leverages differential propagations with zero probability, the ZC attack utilizes linear hulls with zero-correlation as a distinguishing feature.

Developing an ID attack, like many statistical attacks on block ciphers, involves two key phases: identifying a distinguisher and extending it for key recovery. Biham *et al.* [82] introduced not only the ID attack but also a technique to find ID distinguishers known as the *miss-in-the-middle* approach. This approach aims to identify input and output differences that, when propagated forward and backward through the cipher, respectively, contradict each other in the middle. A similar strategy is used in the ZC attack. While the miss-in-the-middle method provides a systematic way to check whether a given input/output difference results in an ID distinguisher, it does not offer a systematic way to choose input and output differences that efficiently result in an ID distinguisher. In practice, one should try several input/output differences (typically with very few active words or bits) and propagate them

halfway with probability one to see if they contradict each other. While this approach might be easy to apply at the word level for strongly aligned and symmetric designs like AES [88] and CLEFIA [89], it is not straightforward to apply it to designs like SKINNY [90] with its slower and less regular diffusion properties. Its application to bit-oriented designs like SIMON, SPECK [14], and Simeck [19] is even more complicated.

Finding ID distinguishers requires tracking the differential propagations through the building blocks of block ciphers at the level of words (nibbles or bytes) or sometimes bits. Regarding the key recovery phase, the attacker has to extend the distinguisher, possibly at two ends, considering more cryptographic properties. This includes identifying the internal states and, subsequently, the key bits whose values are needed to determine the input/output difference of the ID distinguisher. The distinguisher and key recovery parameters, like the number of pairs, should be chosen to minimize the total time complexity of the attack. Overall, building the ID attack is a combinatorial optimization problem that can be daunting and prone to human error if done manually. Therefore, several efforts have been made to automate ID attacks. There are two common approaches to automating the cryptanalytic techniques in symmetric-key cryptography: the first relies on dedicated algorithms, and the second relies on general-purpose solvers. In the alternative method, the cryptanalyst formulates the cryptanalytic challenge either as a Constraint Satisfaction Problem (CSP) or as a Constraint Optimization Problem (COP). Subsequently, they employ advanced general-purpose constraint programming (CP) solvers to resolve it. Note that CP solvers include many solvers, such as Satisfiability Modulo Theories (SMT), Satisfiability (SAT), and MILP solvers.

The first few efforts to automate the ID attack relied on dedicated algorithms and only focused on finding the distinguishers. Examples include the $\mathcal{U}$-method [91] and the UID-method [92]. Another tool based on a dedicated algorithm is the tool provided by Derbez and Fouque at CRYPTO 2016 [93] for finding $\mathcal{DS}$-MITM attacks that is also applicable for finding ID attacks. The main downside of the dedicated algorithms is that they are typically designed for a specific type of design, and modifying them for a new design may require substantial effort. Additionally, providing dedicated, efficient algorithms for solving cryptanalytic problems is typically a challenging task.

At EUROCRYPT 2017, Sasaki and Todo [94] introduced a pioneering approach within the realm of general-purpose solvers, utilizing MILP to identify ID distinguishers. Cui *et al.* [95] also independently introduced this method almost simultaneously and applied it to finding ZC distinguishers. The main advantage of the CP-based method by Sasaki and Todo (and Cui *et al.*) is that the attacker does not have to predict the contradiction mechanism, and this tool can find more complicated contradictions that are not simply detectable by a naive miss-in-the-middle approach. However, the main disadvantage of this approach is that the input/output difference (or linear mask) should be fixed on each try, checking whether the resulting model is unsatisfiable. If so, the input/output difference (resp. linear mask) yields an impossible differential (resp. zero-correlation linear hull). Thus, the attacker needs to try many input/output differences before finding the distinguisher, with non-negligible complexity per try. As a result, the search space for the input/output differences is typically limited to the input/output differences with very few active words or bits. More importantly, the CP-based method by Sasaki and Todo [94] and Cui *et al.* [95] is based on the unsatisfiability of the CP/MILP model and thus cannot be extended to a

unified constraint optimization model for key recovery. This limitation restricts the usage of this CP-based method to only finding the distinguishers.

During EUROCRYPT 2023, Hadipour *et al.* [12] presented a CP-based approach focused on satisfiability within their model, which can be expanded into a unified constraint optimization model for key recovery. This approach is also applicable to discovering ZC and integral attacks. However, the CP model in [12] was word-oriented, suitable for strongly/-moderately aligned designs like SKINNY. At ToSC 2024, Hadipour *et al.* [13] improved upon this technique by broadening its scope to include a bit-wise model that incorporates the internal characteristics of S-boxes. Consequently, they developed a bit-wise CP model rooted in satisfiability to detect ID/ZC distinguishers in designs with weak alignment, such as Ascon [96]. While the methods in [12], [13] improved the best-known ID/ZC and integral attacks on several SPN ciphers, their application to an essential category of block ciphers, i.e., Addition-Rotation-XOR (ARX) and And-Rotation-XOR (AndRX) designs, was left for future work. Additionally, the method introduced in [12], [13] identifies contradictions at the junction of two deterministic propagations (referred to as *direct* contradictions). However, for some ID distinguishers, the contradiction is more complicated and cannot be identified solely based on checking the existence of direct contradictions [97]. We refer to these contradictions as *indirect* contradictions. Therefore, extending the method in [12], [13] to identify indirect contradictions while maintaining the model based on satisfiability is an open problem.

Our contributions. This chapter extends the methods proposed at EUROCRYPT 2023 and ToSC 2024 for finding ID attacks [12], [13] from different aspects. First, we provide a CP-based model based on satisfiability to find ID distinguishers for ARX and AndRX ciphers. Then, as the main contribution regarding the distinguisher part, we propose a CP-based model based on satisfiability capable of identifying particular indirect contradictions for the first time. The application of our CP model for identifying indirect contradictions is not limited to ARX and AndRX ciphers; it applies to other categories of block ciphers like SPN and Feistel ciphers. We also show the applicability of our new CP models for finding ZC distinguishers. Next, we show how to extend the CP-model for key recovery in [12], [13] for bit-wise designs, particularly AndRX designs. Lastly, we put our new models for distinguisher and key-recovery parts into a unified CP model for finding the complete ID attacks, including the key recovery evaluations. To show the usefulness of our methods, we apply them to find ID distinguishers/attacks on several ARX (SPECK [14], ChaCha [15], Chaskey [16], SipHash [17], and LEA [18]) and AndRX (SIMON [14], and Simeck [19]) ciphers and improve the best previous results. Table 4.1, and Table 4.2 provide a summary of our ID distinguishers of ARX ciphers, and the summary of the complete ID attack on AndRX ciphers, discovered by our tool, respectively. Additionally, Table 4.3, and Table 4.4 present an overview of existing attacks (excluding ID attacks) on SIMON, and Simeck, respectively.

- We provide ID distinguishers for ChaCha [15], Siphash [17], SPECK-96, and SPECK-128 [14] for the first time.
- We provide several new ID distinguishers for Chaskey [16] and SPECK with truncated input/output differences.

- We improve ID attacks on SIMON-64-96, SIMON-64-128, SIMON-128-128, and SIMON-128-256 by one round, and SIMON-128-192 by two rounds.
- We provide improved attacks on various variants of SIMON and Simeck: While many previous attacks required the full code-book, we provide ID attacks for the same number of rounds with a lower data complexity than the full code-book.

Table 4.1: ID Distinguishers on ARX ciphers. #R: Length of the distinguisher. #Dist. : Number of distinguishers found using our tool.

Cipher	Contradiction	#R	#Dist.	Ref.
SPECK-32	Direct	6	3	[98]
	Direct	6	$\mathbf{2^4}$	This work
SPECK-48	Direct	6	20	[98]
	Direct	6	$\mathbf{2^{17}}$	This work
SPECK-64	Direct	6	157	[98], [99]
	Direct	6	$\mathbf{2^{33}}$	This work
SPECK-96	Direct	**6**	$\mathbf{2^{65}}$	This work
SPECK-128	Direct	**6**	$\mathbf{2^{97}}$	This work
LEA	Direct	10	-	[95]
	Direct	10	$\mathbf{2^2}$	This work
ChaCha	Direct	**5**	$\mathbf{2^{80}}$	This work
SipHash	Direct	**4**	$\mathbf{2^{14}}$	This work
Chaskey	Direct	4	15	[100]
	Direct	4	$\mathbf{2^7}$	This work

Performance. Unlike previous tools based on unsatisfiability, our tool[1] efficiently identifies a group of ID/ZC distinguishes (or truncated distinguishers) in just one execution, without fixing input/output differences, and terminates within minutes to a few hours on a laptop (Intel Core i5-8250U CPU $1.6GHz \times 8$ and 8GB of memory). The results of our work, along with a comparison to previous works, are presented in Table 4.1 and Table 4.2. MiniZinc [101] is used to model and solve CSP problems.

Outline. We start with an overview of Hadipour *et al.* 's model in Section 4.1. Next, in Section 4.2, we describe our new approach for identifying ID/ZC distinguishers with indirect contradictions for ARX and AndRX designs. Section 4.3 extends our improved model to key-recovery ID attacks on AndRX ciphers. Then we discuss the application of our methods in Section 4.4. Finally, Section 4.5 summarizes the chapter.

[1] https://github.com/Debasmita-isi/zeroplusplus

Table 4.2: Summary of our ID attacks. Dist. = Length of the distinguisher. #R = Number of rounds attacked. † : Distinguisher based on indirect contradiction.

Cipher	Dist.	#R	Time	Data	Mem.	Ref.
SIMON-32-64	11	19	$2^{62.56}$	2^{32}	2^{44}	[44]
	11	19	$2^{58.919}$	2^{32}	$2^{49.674}$	[102]
	11	20	$2^{62.8}$	2^{32}	$2^{43.5}$	[93]
	11	19/20	$\mathbf{2^{59}}$/ $\mathbf{2^{62}}$	$\mathbf{2^{30.79}}$/$\mathbf{2^{31.47}}$	$2^{47.68}$ / $2^{44.48}$	This work
SIMON-48-72	12	20	$2^{70.69}$	2^{48}	2^{58}	[44]
	12	20	$2^{71.278}$	2^{48}	$2^{63.393}$	[102]
	12	20	$\mathbf{2^{67.37}}$	$\mathbf{2^{46.48}}$	2^{64}	This work
SIMON-48-96	12	21	$2^{94.73}$	2^{48}	2^{70}	[44]
	12	21	$2^{94.556}$	2^{48}	$2^{86.447}$	[102]
	12	21	$\mathbf{2^{88.47}}$	$\mathbf{2^{45.48}}$	$2^{76.49}$	This work
SIMON-64-96	13	21	$2^{94.56}$	2^{64}	2^{60}	[44]
	13	21	$2^{95.279}$	2^{64}	$2^{72.469}$	[102]
	13	21/**22**	$\mathbf{2^{86}}$/$\mathbf{2^{93.37}}$	$\mathbf{2^{61.79}}$/$\mathbf{2^{62.37}}$	$2^{68.73}$/$\mathbf{2^{84.4}}$	This work
SIMON-64-128	13	22	$2^{126.56}$	2^{64}	2^{75}	[44]
	13	22	$2^{125.115}$	2^{64}	$2^{98.773}$	[102]
	13	22/**23**	$\mathbf{2^{103}}$/$\mathbf{2^{124}}$	$\mathbf{2^{61.12}}$/$\mathbf{2^{62.47}}$	$2^{86.11}$/$\mathbf{2^{99.5}}$	This work
SIMON-96-96	16	24	$2^{94.62}$	2^{94}	2^{61}	[44]
	16	24	$\mathbf{2^{92}}$	$\mathbf{2^{92.47}}$	$2^{69.39}$	This work
SIMON-96-144	16	25	$2^{142.59}$	2^{96}	2^{77}	[44]
	16	25	$\mathbf{2^{124.793}}$	$\mathbf{2^{94.793}}$	$2^{84.785}$	This work
SIMON-128-128	19	27	$2^{126.6}$	2^{94}	2^{61}	[44]
	19	**28**	$\mathbf{2^{114.641}}$	$\mathbf{2^{110.6}}$	$\mathbf{2^{86}}$	This work
SIMON-128-192	19	28	$2^{190.56}$	2^{128}	2^{77}	[44]
	19	**29/30**	$\mathbf{2^{167}}$/$\mathbf{2^{181}}$	$\mathbf{2^{127.11}}$/$\mathbf{2^{127.64}}$	$\mathbf{2^{97.12}}$/$\mathbf{2^{112.68}}$	This work
SIMON-128-256	19	30	$2^{254.68}$	2^{128}	2^{111}	[44]
	19	30/**31**	$\mathbf{2^{232}}$/$\mathbf{2^{248}}$	$\mathbf{2^{125.05}}$/$\mathbf{2^{125.47}}$	$2^{124.1}$/$\mathbf{2^{127.5}}$	This work
Simeck-32	11	20	$2^{61.11}$	2^{32}	2^{51}	[103]
	11	20	$\mathbf{2^{55.79}}$	$\mathbf{2^{29.79}}$	$\mathbf{2^{50.79}}$	This work
Simeck-48	15†	25	$2^{94.23}$	2^{46}	2^{67}	[103]
	15†	25	$\mathbf{2^{93.05}}$	$2^{47.05}$	$2^{68.12}$	This work
Simeck-64	17†	27	$2^{126.56}$	2^{63}	2^{68}	[103]
	17†	27	$\mathbf{2^{126}}$	$2^{63.47}$	$2^{68.45}$	This work

Table 4.3: Cryptanalysis of SIMON

Cipher	Attack	#R	Time	Data	Mem.	Ref.
SIMON-32-64	Differential	22	$2^{58.76}$	2^{32}	-	[104]
	Linear	23	$2^{61.84}$	$2^{31.19}$	-	[105]
	MITM	18	$2^{62.57}$	2^{3}	-	[106]
	DS-MITM	16	$2^{56.29}$	2^{30}	-	[107]
	Zero-correlation	21	$2^{59.4}$	2^{32}	2^{31}	[108]
	Integral	24	2^{63}	2^{32}	$2^{33.64}$	[109]
SIMON-48-72	Differential	23	$2^{63.25}$	2^{47}	-	[110]
	Linear	24	$2^{67.89}$	$2^{47.92}$	-	[105]
	MITM	17	$2^{71.75}$	2^{3}	-	[106]
	DS-MITM	16	$2^{63.24}$	2^{44}	-	[107]
	Zero-correlation	21	$2^{59.4}$	2^{48}	2^{43}	[108]
	Integral	24	2^{71}	2^{48}	2^{50}	[109]
SIMON-48-96	Differential	24	$2^{78.99}$	2^{48}	-	[110]
	Linear	25	$2^{47.92}$	$2^{89.89}$	-	[105]
	MITM	19	$2^{95.26}$	2^{3}	-	[106]
	DS-MITM	18	$2^{91.62}$	2^{45}	-	[107]
	Zero-correlation	22	$2^{80.5}$	2^{48}	2^{43}	[108]
	Integral	25	2^{95}	2^{48}	2^{50}	[109]
SIMON-64-96	Differential	30	2^{88}	$2^{63.3}$	-	[110]
	Linear	30	$2^{93.62}$	$2^{63.53}$	-	[105]
	MITM	17	$2^{94.05}$	2^{3}	-	[106]
	DS-MITM	18	$2^{95.94}$	2^{58}	-	[107]
	Zero-correlation	23	$2^{90.4}$	2^{64}	2^{54}	[108]
SIMON-64-128	Differential	31	2^{120}	$2^{63.3}$	-	[110]
	Linear	31	2^{120}	$2^{63.53}$	-	[105]
	MITM	19	$2^{126.01}$	2^{3}	-	[106]
	DS-MITM	19	$2^{100.94}$	2^{58}	-	[107]
	Zero-correlation	24	$2^{116.8}$	2^{64}	2^{54}	[108]
SIMON-96-96	Differential	37	$2^{87.17}$	2^{95}	-	[110]
	Linear	43	$2^{89.6}$	2^{94}	-	[111]
	MITM	-	-	-	-	-
	DS-MITM	18	$2^{77.92}$	2^{36}	-	[107]
	Zero-correlation	-	-	-	-	-
SIMON-96-144	Differential	37	$2^{130.75}$	2^{95}	-	[110]
	Linear	45	$2^{136.5}$	2^{95}	-	[111]
	MITM	21	$2^{141.27}$	2^{4}	-	[106]
	DS-MITM	20	$2^{111.90}$	2^{36}	-	[107]
	Zero-correlation	28	2^{141}	2^{96}	2^{85}	[108]
SIMON-128-128	Differential	50	$2^{119.19}$	2^{127}	-	[110]
	Linear	53	2^{121}	2^{127}	-	[111]
	MITM	-	-	-	-	-
	DS-MITM	21	$2^{106.98}$	2^{47}	-	[107]
	Zero-correlation	-	-	-	-	-
SIMON-128-192	Differential	51	$2^{183.17}$	2^{127}	-	[110]
	Linear	55	$2^{185.2}$	2^{127}	-	[111]
	MITM	25	$2^{190.60}$	2^{3}	-	[106]
	DS-MITM	23	$2^{150.46}$	2^{62}	-	[107]
	Zero-correlation	32	$2^{156.8}$	2^{128}	2^{117}	[108]
SIMON-128-256	Differential	51	$2^{247.17}$	2^{127}	-	[110]
	Linear	56	2^{249}	2^{126}	-	[111]
	MITM	25	$2^{253.94}$	2^{3}	-	[106]
	DS-MITM	26	$2^{250.08}$	2^{53}	-	[107]
	Zero-correlation	34	$2^{255.6}$	2^{128}	2^{117}	[108]

Table 4.4: Cryptanalysis of Simeck

Cipher	Attack	#R	Time	Data	Mem.	Ref.
Simeck-32-64	Differential	22	$2^{57.9}$	2^{32}	-	[104]
	Linear	23	$2^{61.78}$	$2^{31.91}$	-	[112]
	Zero-correlation	21	$2^{58.78}$	2^{32}	2^{31}	[113]
	Integral	22	2^{63}	2^{31}	$2^{55.88}$	[114]
Simeck-48-96	Differential	28	$2^{68.3}$	2^{46}	-	[104]
	Linear	32	$2^{90.9}$	2^{47}	-	[111]
	Zero-correlation	27	$2^{85.67}$	2^{48}	$2^{58.78}$	[97]
	Integral	26	2^{95}	2^{47}	$2^{82.52}$	[114]
Simeck-64-128	Differential	35	$2^{116.3}$	2^{63}	-	[104]
	Linear	42	$2^{123.9}$	$2^{63.5}$	-	[111]
	Zero-correlation	31	$2^{124.08}$	2^{64}	$2^{89.35}$	[97]
	Integral	30	$2^{127.3}$	2^{63}	$2^{109.02}$	[114]

4.1 Previous CP Model for Deterministic Trails

Here, we revisit the bit-wise CP model discussed in [13], [115], which encapsulates the propagation of deterministic differential and linear trails. We explain the model for differential trails, but a similar approach can be used for linear trails. The idea is to encode the difference at each bit position via an integer variable with a $\{-1, 0, 1\}$ domain. The integer values "0" and "1" represent the fixed difference value of "0" and "1", and "-1" means the difference value is either "0" or "1" (i.e., unknown). Then, deterministic differential trails through `XOR`, `Branching`, and S-boxes can be encoded in the following manner.

Proposition 4.1 (`Branching` **[13], [115])** *For* $f : \mathbb{F}_2 \to \mathbb{F}_2^n$, $f(x) = (y_0, y_1, \ldots, y_{n-1})$ *where* $y_0 = y_1 = \cdots = x$, *the valid deterministic differential propagations satisfy*

$$\texttt{Branch}(\texttt{x}, \texttt{y}_0, \ldots, \texttt{y}_{n-1}) := \bigwedge_{i=0}^{n-1} (\texttt{y}_i = \texttt{x}),$$

where the integer variables $\texttt{x}, \texttt{y}_i \in \{-1, 0, 1\}$ *encode the difference in* x, y_i *for* $0 \leqslant i \leqslant n-1$.

Proposition 4.2 (`XOR` **[13], [115])** *For* $f : \mathbb{F}_2^n \to \mathbb{F}_2$, $f(x_0, x_1, \ldots, x_{n-1}) = y$, *where* $y = x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1}$, *the valid deterministic differential propagations satisfy*

$$\texttt{XOR}(\texttt{y}, \texttt{x}_0, \ldots, \texttt{x}_{n-1}) := \begin{cases} \texttt{if } \bigvee_{i=0}^{n-1} (\texttt{x}_i = -1) \texttt{ then y} = -1 \\ \texttt{else y} = \texttt{x}_0 + \cdots + \texttt{x}_{n-1} \mod 2 \end{cases}$$

The propagation of deterministic differential/linear trails through S-boxes can be explained as follows. We can model this by using the *Difference Distribution Table* (DDT) of the S-box, through which we can identify the differential propagations that have a known output difference in at least one bit position. These propagations are known as bit-wise deterministic differential propagations. To be more precise, let us assume that the S-box is an $m \times n$ S-box. We examine all input activeness patterns in $\{-1, 0, 1\}^m$, where for each input activeness pattern, we check whether at least one bit of the output difference is known to be "0" or "1" with certainty. Next, we model all deterministic bit-wise differential propagations through the S-box using CP constraints. For details, we refer to [13, Sec 3.2]. A similar

method works for modeling the bit-wise deterministic linear trails of S-boxes, using the *Linear Approximation Table* (LAT) to identify the bit-wise deterministic linear propagations. The CP constraints for bit-wise deterministic differential/linear propagation through S-boxes can be automatically derived with an extended version [13], [115] of the S-box Analyzer tool [116]. In Subsection 4.2.1, we extend this method to model the building blocks of AndRX and ARX ciphers, particularly the modular addition.

4.1.1 CP Model for Finding ID/ZC Distinguishers

The method for identifying ID (resp. ZC) distinguishers in [12], [13] utilizes the *miss-in-the-middle* technique from [82]. According to this technique, we propagate a given input and output differences (resp. linear masks) through the block cipher with certainty forward and backward, respectively. If the two propagations contradict each other somewhere in the middle, then we can prove that the given input difference (resp. linear mask) never propagates to the given output difference (resp. linear mask). As a result, we have an ID (resp. ZC) distinguisher. The idea of Hadipour *et al.* is to model the deterministic differential (resp. linear) propagations through the block cipher in two opposite directions using CP constraints. The CP model is then extended by including some contradiction checker constraints for each bit position to guarantee the contradiction between the two deterministic propagations in at least one bit position. This way, any feasible solutions of the CP model are an impossible differential (resp. zero-correlation) distinguisher. The main advantage is that there are no constraints for the input/output differences (resp. linear masks), and the CP model is based on satisfiability. For more details, please refer to [12], [13].

This method only identifies ID/ZC distinguishers relying on *direct* contradictions, i.e., contradictions that happen at the junction of two deterministic differential (or linear) trails propagated in two opposite directions. However, some ID distinguishers [97] are not detectable by only checking the existence of direct contradictions. The contradictions in these distinguishers are more complicated, and we refer to them as *indirect* contradictions. To address this gap, in Subsection 4.2.2, we provide a new CP model based on satisfiability, which is capable of identifying a particular type of indirect contradiction.

4.1.2 Unified CP Model for Finding Complete ID Attacks

Once we have a CP model based on satisfiability, we can extend it to find an optimal complete ID key recovery attack. We briefly recall the general view of the first CP model for finding complete ID attacks in [12]. As visualized in Figure 2.8, assume that we split the block ciphers E into three sub-ciphers $E = E_{\text{F}} \circ E_{\text{D}} \circ E_{\text{B}}$, such that the distinguisher covers E_{D}, E_{B}, and E_{F} denote the extension of the distinguisher backwards and forwards for key recovery, respectively. Also, assume that $\texttt{CSP}_{\text{D}}$ represents the Constraint Satisfaction Problem (CSP) modeling the distinguisher part. The idea is to extend $\texttt{CSP}_{\text{D}}$ by additional CP variables/constraints that aim at modeling the key recovery procedure, as well as the complexity analysis of ID key recovery.

The key recovery process involves first propagating the input/output difference of the ID distinguisher backward/forward, then identifying the filters, and finally, pinpointing the

cell/bit positions within the internal state or sub-keys whose difference or value is necessary for the guess-and-filter step. Once we have this information, we can provide a rough estimation of the time, memory, and data complexity of the ID attack, along with a sketch of the key recovery procedure. For this purpose, according to [12], one can define four types of binary CP variables for the extended parts E_{B} and E_{F}. The first type of binary variable encodes whether the difference in a particular position through E_{B} or E_{F} is zero. The second type of variable encodes whether a particular position acts as a filter. The third variable type encodes whether the difference of a certain cell within the internal state or sub-keys should be known, and the fourth variable type encodes whether the value pair at a certain cell should be known. Next, we can define some constraints on these variables to model the guess-and-filter procedure and the complexity formula of ID key recovery. Finally, one can integrate the CP models for key-bridging [117] into this model to consider the relation between the involved keys $k_{\text{B}} \cup k_{\text{F}}$ (see Figure 2.8) to identify the actual size of $k_{\text{B}} \cup k_{\text{F}}$, which is a critical parameter in the time complexity of ID key recovery.

4.2 Modeling the Distinguishers

In this section, we expand the bit-wise CP model presented in [13] for identifying ID/ZC distinguishers in two key aspects. First, we introduce a rule to encode `AND` and modular addition operations within the bit-wise CP model. This enables us to create a CP model based on satisfiability to find ID/ZC distinguishers for ARX and AndRX ciphers. Subsequently, and of greater significance, we extend the bit-wise model to detect ID/ZC distinguishers with more intricate contradictions beyond direct ones. This adaptation empowers our new model to identify the longest existing ID/ZC distinguishers of Simeck that are not detectable by the models in [12], [13]. The versatility of our new model in identifying complex contradictions is not restricted to ARX and AndRX ciphers; it can also be applied to other designs, such as SPN ciphers. Similar to the CP models in [12], [13], the primary advantage of our new model is its extensibility to a unified optimization problem for discovering a complete ID attack.

4.2.1 Modeling the Distinguishers for ARX and AndRX Ciphers

Here, we propose some rules to model deterministic differential (linear) propagation through `AND` and modular addition operations. We elaborate on our modeling of deterministic differential trails, noting that the same approach applies to linear trails.

Proposition 4.3 (`AND`) *Let $f : \mathbb{F}_2^n \to \mathbb{F}_2$ be such that $y = f(x_0, x_1, \ldots, x_{n-1}) = \bigwedge_{i=0}^{n-1} x_i$. Let $\mathtt{x}_i$ and $\mathtt{y}$ be the corresponding integer variables with domain $\{-1, 0, 1\}$ to encode the difference in x_i and y. Then, the valid deterministic differential propagations can be represented by satisfying*

$$\texttt{AND}\,(\mathtt{x}_0, \mathtt{x}_1, \ldots, \mathtt{x}_{n-1}, \mathtt{y}) \coloneqq \begin{cases} \texttt{if} & \mathtt{x}_0 = \mathtt{x}_1 = \ldots = \mathtt{x}_{n-1} = 0 \;\; \texttt{then} \;\; \mathtt{y} = 0 \\ \texttt{else} & \mathtt{y} = -1 \end{cases}$$

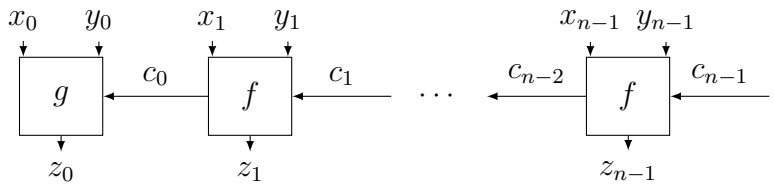


Figure 4.1: Representing the modular addition $X \boxplus Y$ using full-adders f and a half-adder g.

Suppose that $f : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ is such that $z = f(x, y) = x \boxplus y$, where $\boxplus$ denotes modular addition modulo 2^n. Assume that we represent x as a bit-vector $x_0||x_1|| \cdots ||x_{n-1}$, where $x_i \in \mathbb{F}_2$ for $0 \leqslant i \leqslant n-1$, and x_0 is the Most Significant Bit (MSB). As visualized in Figure 4.1, we decompose the modular addition into n smaller Boolean functions (a.k.a. full-/half-adders). Assuming that c_i for $0 \leqslant i \leqslant n-1$ are binary variables to represent the carry bits, we define $(z_i, c_i) = f(x_i, y_i, c_{i+1}) \coloneqq (x_i \oplus y_i \oplus c_i,\ x_{i+1} \cdot y_{i+1} \oplus c_{i+1} \cdot (x_{i+1} \oplus y_{i+1}))$ for $1 \leqslant i \leqslant n-1$, where $c_{n-1} = 0$. Additionally, we define $z_0 = g(x_0, y_0, c_0) \coloneqq x_0 \oplus y_0 \oplus c_0$. Next, we encode the propagation of deterministic differential trails through f and g using CP constraints. As for g, we can apply the principles outlined for modeling `XOR` in Proposition 4.2. Regarding f, adopting a methodology akin to that detailed in [13], we treat it as an S-box. By referencing its *Differential Distribution Table* (DDT), we pinpoint differential propagations where the difference is definitively known in at least one output bit. These are termed deterministic bit-wise differential propagations and are expressed using CP constraints. Proposition 4.4 and Proposition 4.5 briefly describe our CP constraints to model modular addition.

Proposition 4.4 (`Full adder`) *Assume that* $(z, c') = f(x, y, c) = (x \oplus y \oplus c, x \cdot y \oplus c \cdot (x \oplus y))$, *and let* $\mathtt{x}, \mathtt{y}, \mathtt{c}, \mathtt{z}, \mathtt{c}'$ *denote the integer variables with domain* $\{-1, 0, 1\}$ *to encode the corresponding differences. Then, the following CP constraints model all valid bit-wise deterministic differential propagations through the full adder:*

$$\mathtt{FA}(\mathtt{x}, \mathtt{y}, \mathtt{c}, \mathtt{z}, \mathtt{c}') \coloneqq \begin{cases} \mathtt{if} & (\mathtt{x} = 0 \wedge \mathtt{y} = 0 \wedge \mathtt{c} = 0) & \mathtt{then}\ (\mathtt{z} = 0 \wedge \mathtt{c}' = 0) \\ \mathtt{elseif} & (\mathtt{x} = 0 \wedge \mathtt{y} = 0 \wedge \mathtt{c} = 1) & \mathtt{then}\ (\mathtt{z} = 1 \wedge \mathtt{c}' = -1) \\ \mathtt{elseif} & (\mathtt{x} = 0 \wedge \mathtt{y} = 1 \wedge \mathtt{c} = 0) & \mathtt{then}\ (\mathtt{z} = 1 \wedge \mathtt{c}' = -1) \\ \mathtt{elseif} & (\mathtt{x} = 0 \wedge \mathtt{y} = 1 \wedge \mathtt{c} = 1) & \mathtt{then}\ (\mathtt{z} = 0 \wedge \mathtt{c}' = -1) \\ \mathtt{elseif} & (\mathtt{x} = 1 \wedge \mathtt{y} = 0 \wedge \mathtt{c} = 0) & \mathtt{then}\ (\mathtt{z} = 1 \wedge \mathtt{c}' = -1) \\ \mathtt{elseif} & (\mathtt{x} = 1 \wedge \mathtt{y} = 0 \wedge \mathtt{c} = 1) & \mathtt{then}\ (\mathtt{z} = 0 \wedge \mathtt{c}' = -1) \\ \mathtt{elseif} & (\mathtt{x} = 1 \wedge \mathtt{y} = 1 \wedge \mathtt{c} = 0) & \mathtt{then}\ (\mathtt{z} = 0 \wedge \mathtt{c}' = -1) \\ \mathtt{elseif} & (\mathtt{x} = 1 \wedge \mathtt{y} = 1 \wedge \mathtt{c} = 1) & \mathtt{then}\ (\mathtt{z} = 1 \wedge \mathtt{c}' = 1) \\ \mathtt{else} & & (\mathtt{z} = -1 \wedge \mathtt{c}' = -1) \end{cases}$$

Proposition 4.5 (`Modular Addition`) *Assume we express the modular addition* $z = x \boxplus y$ *as a composition of* $n-1$ *full adders* f *along with a half adder* g *as explained before, and let* $\mathtt{x}_i, \mathtt{y}_i, \mathtt{z}_i, \mathtt{c}_i$ *denote the integer variables with the domain* $\{-1, 0, 1\}$ *for the corresponding difference at bit positions* x_i, y_i, z_i, c_i. *Then the following constraints model the bit-wise*

deterministic differential propagations through modular addition:

$$\texttt{ModAdd} := \left(\bigwedge_{i=1}^{n-1} \texttt{FA}(\texttt{x}_i, \texttt{y}_i, \texttt{c}_i, \texttt{z}_i, \texttt{c}_{i-1}) \right) \wedge \texttt{XOR}(\texttt{z}_0, \texttt{x}_0, \texttt{y}_0, \texttt{c}_0,) \wedge (\texttt{c}_{n-1} = 0). \tag{4.1}$$

To model bit-wise deterministic linear propagations, we follow a similar approach. In this case, concerning the vectorial Boolean functions f, g, we utilize their *Linear Approximation Tables* (LAT).

4.2.2 New CP Model to Identify Indirect Contradictions

We now provide a CP model based on satisfiability, which can identify both direct and indirect contradictions. In particular, we focus on the indirect contradictions first described in [97]. We first give the intuition for our approach. Assume no direct contradiction exists between the two deterministic differential trails propagated in opposite directions. However, what if we merge the information from the two deterministic propagations at a particular round and propagate this new information with probability one in both directions such that further propagation of this new information contradicts one of the original forward and backward propagations? If so, based on the proof by contradiction, we can conclude that the original two deterministic differential trails cannot exist simultaneously. Let us take a very simple and abstract example to explain the idea. Assume we are analyzing a cipher with two differential trails:

- **Forward Trail:** Starts with an input difference Δx and propagates it through the rounds to an output difference Δy.
- **Backward Trail:** Starts with an output difference $\Delta y'$ and propagates it back to an input difference $\Delta x'$ (in reverse).

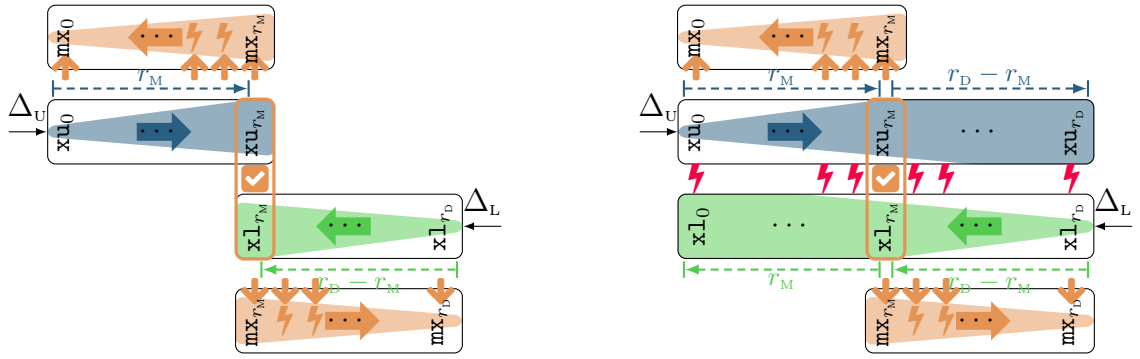
Now, imagine that there are no contradictions between the two trails when considered separately (i.e., no direct contradictions exist). Now, let us consider merging the information from both trails at a particular round, say round r. We combine the information from both trails and propagate it with certainty (probability one), meaning that whatever information we have at this round must propagate exactly in both directions (forward and backward).

- **Step 1: Merge Information.** We combine the two trails' information at round r. Now, let us say the forward trail suggests a certain difference Δz at this round, and the backward trail suggests a different difference Δw at the same round.
- **Step 2: Propagate with certainty.** After merging the two sets of information, we propagate this new combined information forward and backward in both directions with certainty. Now, this new forward (resp. backward) propagation of this new information contradicts with the original backward (resp. forward) propagation. Specifically, we find that the newly propagated information does not match with the original deterministic trails at any of the subsequent rounds.

In cryptanalysis, contradictions are powerful because they imply that a certain scenario (in this case, the existence of both trails) is impossible. The process of propagating the combined information with certainty effectively rules out the possibility that both trails can coexist, even though they might not have contradicted each other directly (no direct contradiction).

To include these cases in our CP model, we extend it with new CP constraints that merge the information from both deterministic differential trails and then propagate the latest information with probability one in both directions. We equip this CP model with extra contradiction checkers between the original and new differential trails. Lastly, we include a constraint to ensure that at least one of the (direct or indirect) contradiction checkers is activated.

Modeling indirect contradictions. We detect an ID or ZC characteristic for r_{D} rounds of a block cipher E (denoted as E_{D}) by dividing E_{D} into two parts: E_{U} covering r_{M} rounds, and E_{L} covering the remaining $(r_{\mathrm{D}} - r_{\mathrm{M}})$ rounds. The trails identified in E_{U} are termed the upper trail, while those in E_{L} are referred to as the lower trail. After r rounds, the internal state of E_{U} is denoted as xu_r for $0 \leqslant r \leqslant r_{\mathrm{M}}$, and correspondingly, for E_{L}, it is $xl_{r_{\mathrm{D}}-r}$ for $0 \leqslant r \leqslant (r_{\mathrm{D}} - r_{\mathrm{M}})$. Therefore, $xu_{r_{\mathrm{M}}}$ and $xl_{r_{\mathrm{M}}}$ correspond to the same internal state at the junction of the two sub-ciphers.



(a) Indirect contradiction model

(b) Combined model

Figure 4.2: Model for impossible-differential distinguishers with indirect contradiction.

Let $\mathtt{xu}_r$ and $\mathtt{xl}_r$ represent the difference patterns of the state variables xu_r and xl_r, respectively, as illustrated in Figure 4.2. In particular, $\mathtt{xu}_r[i]$ (or $\mathtt{xl}_r[i]$) is an integer variable with a domain of $\{-1, 0, 1\}$, depicting the difference pattern in the i-th bit of xu_r (or xl_r). We represent the propagation of the deterministic truncated differential path via E_{U} and E_{L} in both the encryption (forward) and decryption (backward) directions using distinct CSP models. For this purpose, we utilize the propagation rules from [12], [13] along with our new rules from Subsection 4.2.1. We denote the model for the propagation of deterministic truncated trails through E_{U} and E_{L}^{-1} as $\mathtt{CSP}_{\mathrm{U}}(\mathtt{xu}_0, \ldots, \mathtt{xu}_{r_{\mathrm{M}}})$, and $\mathtt{CSP}_{\mathrm{L}}(\mathtt{xl}_{r_{\mathrm{M}}}, \ldots, \mathtt{xl}_{r_{\mathrm{D}}})$, respectively. Also, let f denote the round function of block cipher E. We represent the CP constraints for the propagation of deterministic truncated trails over f (resp. f^{-1}) as $\mathtt{f}_{\mathrm{U}}(\mathtt{x}, \mathtt{y})$ (resp. $\mathtt{f}_{\mathrm{L}}(\mathtt{y}, \mathtt{x})$), where $\mathtt{x}$ (resp. $\mathtt{y}$) denotes the activeness pattern at the input (resp. output) of f.

Now, we explain how we model the merging of information from the upper and lower trails and identify the indirect contradictions by defining some new CP variables and constraints. In round r_{M}, we need to merge the information from the upper and lower trails at the junction of E_{U} and E_{L}. After merging the information in round r_{M}, we need to propagate this new information forward and backward. After each round of propagation of the new information, we check whether the new activeness pattern is consistent with the activeness pattern at the corresponding state from the original propagation. Moreover, to determine the activeness pattern at each round, we must merge the information from the previous round in the new propagation with the information from the corresponding state in the original propagation.

To this end, for the internal state at each round, we define three new types of integer variables $\mathtt{mx}_r$, $\mathtt{mx}'_r$, and $\mathtt{mc}_r$ with a domain $\{-1, 0, 1\}$. The integer variable $\mathtt{mx}'_r$ encodes the information propagated from the previous round, $\mathtt{mx}_r$ encodes the result of merging the information from the previous round with the information from the corresponding state at the original propagation, and $\mathtt{mc}_r$ checks if there is a contradiction between the new propagation and the original one. A more detailed explanation regarding these variables is as follows.

To merge the information at each round of propagation, we first define the predicate $\mathtt{merge}(\mathtt{xu}_{r_{\mathrm{M}}}[i], \mathtt{xl}_{r_{\mathrm{M}}}[i], \mathtt{mx}_{r_{\mathrm{M}}}[i], \mathtt{mc}_{r_{\mathrm{M}}}[i])$:

$$\begin{cases} \mathtt{if} & (\mathtt{xu}_{r_{\mathrm{M}}}[i] = -1) & \mathtt{then}\ (\mathtt{mx}_{r_{\mathrm{M}}}[i] = \mathtt{xl}_{r_{\mathrm{M}}}[i] \ \wedge\ \mathtt{mc}_{r_{\mathrm{M}}}[i] = 0) \\ \mathtt{elseif} & (\mathtt{xl}_{r_{\mathrm{M}}}[i] = -1) & \mathtt{then}\ (\mathtt{mx}_{r_{\mathrm{M}}}[i] = \mathtt{xu}_{r_{\mathrm{M}}}[i] \ \wedge\ \mathtt{mc}_{r_{\mathrm{M}}}[i] = 0) \\ \mathtt{elseif} & (\mathtt{xu}_{r_{\mathrm{M}}}[i] = \mathtt{xl}_{r_{\mathrm{M}}}[i]) & \mathtt{then}\ (\mathtt{mx}_{r_{\mathrm{M}}}[i] = \mathtt{xu}_{r_{\mathrm{M}}}[i] \ \wedge\ \mathtt{mc}_{r_{\mathrm{M}}}[i] = 0) \\ \mathtt{else} & & (\mathtt{mc}_{r_{\mathrm{M}}}[i] = 1) \end{cases}$$

Using this predicate, we start from the meeting point of E_{U} and E_{L} and merge the activeness patterns from the upper and lower trails at the junction of E_{U} and E_{L}:

$$\mathtt{CSP}_{\mathrm{M}}(\mathtt{xu}_{r_{\mathrm{M}}},\ \mathtt{xl}_{r_{\mathrm{M}}},\ \mathtt{mx}_{r_{\mathrm{M}}},\ \mathtt{mc}_{r_{\mathrm{M}}}) \coloneqq \bigwedge_{i=0}^{n-1} \mathtt{merge}(\mathtt{xu}_{r_{\mathrm{M}}}[i],\ \mathtt{xl}_{r_{\mathrm{M}}}[i],\ \mathtt{mx}_{r_{\mathrm{M}}}[i],\ \mathtt{mc}_{r_{\mathrm{M}}}[i]) \tag{4.2}$$

Now, we must propagate the result of merging, namely $\mathtt{mx}_{r_{\mathrm{M}}}$, to both encryption and decryption direction with certainty. We first explain the backward propagation. Assume that we aim to determine the activeness pattern $\mathtt{mx}_{r-1}$ at round $r-1$ based on the activeness pattern $\mathtt{mx}_r$ at round r in the new propagation and also the activeness pattern $\mathtt{xu}_{r-1}$ at round $r-1$ in the original propagation. For this purpose, using $\mathtt{f}_{\mathrm{L}}(\mathtt{mx}_r, \mathtt{mx}'_{r-1})$ we first propagate $\mathtt{mx}_r$ into $\mathtt{mx}'_{r-1}$ backward through f. Next, we merge the activeness patterns at $\mathtt{mx}'_{r-1}$ and $\mathtt{xu}_{r-1}$ into $\mathtt{mx}_{r-1}$, and finally, we check if there is a contradiction between $\mathtt{mx}'_{r-1}$ and $\mathtt{xu}_{r-1}$ using $\mathtt{mc}_{r-1}$. We use the following constraint for this purpose:

$$\begin{aligned} &\mathtt{CSP}_{\mathrm{B}}(\mathtt{xu}_0, \ldots, \mathtt{xu}_{r_{\mathrm{M}}-1}, \mathtt{mx}_0, \ldots, \mathtt{mx}_{r_{\mathrm{M}}},\ \mathtt{mc}_0, \ldots, \mathtt{mc}_{r_{\mathrm{M}}-1}) \coloneqq \\ &\qquad \bigwedge_{r=1}^{r_{\mathrm{M}}} \left(\mathtt{f}_{\mathrm{L}}(\mathtt{mx}_r, \mathtt{mx}'_{r-1}) \wedge \left(\bigwedge_{i=0}^{n-1} \mathtt{merge}_i(\mathtt{xu}_{r-1}[i], \mathtt{mx}'_{r-1}[i], \mathtt{mx}_{r-1}[i], \mathtt{mc}_{r-1}[i]) \right) \right) \end{aligned} \tag{4.3}$$

Similarly, we model the merging and propagation in the forward direction. For each round r, where $r_{\mathrm{M}} \leqslant r < r_{\mathrm{D}}$, we first use $\mathtt{f}_{\mathrm{U}}(\mathtt{mx}_r, \mathtt{mx}'_{r+1})$ to propagate $\mathtt{mx}_r$ into $\mathtt{mx}'_{r+1}$. Next,

we merge $\mathtt{mx}'_{r+1}$ with $\mathtt{xl}_{r+1}$ into $\mathtt{mx}_{r+1}$ and check if there is a contradiction between $\mathtt{mx}_{r+1}$ and $\mathtt{xl}_{r+1}$ using $\mathtt{mc}_{r+1}$. To this end, we use the following constraint:

$$\mathtt{CSP}_{\text{F}}(\mathtt{xl}_{r_{\text{M}}+1},\ldots,\mathtt{xl}_{r_{\text{D}}},\mathtt{mx}_{r_{\text{M}}},\ldots,\mathtt{mx}_{r_{\text{D}}},\ \mathtt{mc}_{r_{\text{M}}+1},\ldots,\mathtt{mc}_{r_{\text{D}}}) := \tag{4.4}$$
$$\bigwedge_{r=r_{\text{M}}}^{r_{\text{D}}}\left(\mathtt{f}_{\text{U}}(\mathtt{mx}_r,\mathtt{mx}'_{r+1})\wedge\left(\bigwedge_{i=0}^{n-1}\mathtt{merge}_i(\mathtt{xl}_{r+1}[i],\mathtt{mx}'_{r+1}[i],\mathtt{mx}_{r+1}[i],\mathtt{mc}_{r+1}[i])\right)\right)$$

We must ensure that at least one of the contradiction checker constraints is met. To achieve this, we introduce the following constraint. It ensures that there is a mismatch between the new backward propagation over E_{U}^{-1} and the original forward propagation over E_{U}, or the new forward propagation over E_{L}, and the original backward propagation over E_{L}^{-1} in at least one bit across the entire distinguisher.

$$\mathtt{CSP}_{\text{C}}(\mathtt{mc}_0,\mathtt{mc}_1,\ldots,\mathtt{mc}_{r_{\text{D}}}) := \bigvee_{r=0}^{r_{\text{D}}-1}\left(\bigvee_{i=0}^{n-1}(\mathtt{mc}_r[i]=1)\right) \tag{4.5}$$

The combination of the aforementioned CSP models denoted as $\mathtt{CSP}_{\text{D}}$, forms an integrated CP model focused on satisfiability. Its feasible solutions correspond to impossible differential distinguishers:

$$\mathtt{CSP}_{\text{D}} := \mathtt{CSP}_{\text{U}}\wedge\mathtt{CSP}_{\text{L}}\wedge\mathtt{CSP}_{\text{M}}\wedge\mathtt{CSP}_{\text{F}}\wedge\mathtt{CSP}_{\text{B}}\wedge\mathtt{CSP}_{\text{C}}$$

Hence, when provided r_{D} and r_{M}, this approach identifies a distinguisher for r_{D} rounds of the block cipher where either we find a contradiction in r_{M} round (in case of direct contradiction), or we can find some indirect contradiction where r_{M} is the round where first time merging happens ($r_{\text{M}}=8$ in Figure 4.4). We detail our model for ID distinguishers, which is similarly applicable to ZC distinguishers. We provide a more detailed analysis of the attack model to identify indirect contradictions, as explained in Figure 4.4 in the subsequent section.

Combined model of indirect and direct contradictions. We can extend our idea to construct a combined CSP model capable of identifying both direct and indirect contradictions. Let $\mathtt{CSP}_{\text{U}}(\mathtt{xu}_0,\mathtt{xu}_1,\ldots,\mathtt{xu}_{r_{\text{D}}})$ and $\mathtt{CSP}_{\text{L}}(\mathtt{xl}_0,\mathtt{xl}_1,\ldots,\mathtt{xl}_{r_{\text{D}}})$ denote the CSP models for the forward and backward propagations through E_{D}, and E_{D}^{-1}, respectively. We extend the models to the full r_{D} rounds. Next, following a similar approach, we can construct the CSP models $\mathtt{CSP}_{\text{M}}$, $\mathtt{CSP}_{\text{B}}$, and $\mathtt{CSP}_{\text{F}}$ as described in Equation 4.2, Equation 4.3, and Equation 4.4, respectively. Lastly, we introduce the following constraints to guarantee the inconsistency among the four aforementioned propagations:

$$\mathtt{CSP}_{\text{C}}(\mathtt{xu}_0,\ldots,\mathtt{xu}_{r_{\text{D}}},\mathtt{xl}_0,\ldots,\mathtt{xl}_{r_{\text{D}}},\mathtt{mc}_0,\mathtt{mc}_1,\ldots,\mathtt{mc}_{r_{\text{D}}}) :=$$
$$\bigvee_{r=0}^{r_{\text{D}}-1}\left(\bigvee_{i=0}^{n-1}(\mathtt{mc}_r[i]=1)\vee(\mathtt{xu}_r[i]+\mathtt{xl}_r[i]=1)\right)$$

The combination of the CSP models mentioned above, denoted as $\mathtt{CSP}_{\text{D}}$, forms an integrated CP model focused on satisfiability, effectively identifying ID distinguishers through direct and indirect contradictions:

$$\mathtt{CSP}_{\text{D}} := \mathtt{CSP}_{\text{U}}\wedge\mathtt{CSP}_{\text{L}}\wedge\mathtt{CSP}_{\text{M}}\wedge\mathtt{CSP}_{\text{F}}\wedge\mathtt{CSP}_{\text{B}}\wedge\mathtt{CSP}_{\text{C}}$$

We apply the above idea to several ARX and AndRX ciphers and discover several new distinguishers. Section 4.4 elaborates on the details of our applications.

4.2.3 Modeling ZC Distinguishers

Both ID and ZC distinguishers primarily leverage the *miss-in-the-middle* technique in their construction. In Subsection 4.2.2, we provided a CP model focused on satisfiability to identify direct and indirect contradictions when searching for ID distinguishers. A similar approach applies to finding ZC distinguishers. However, we encountered specific challenges when developing the CP model for ZC distinguishers based on satisfiability for ARX ciphers. This section delves into these challenges and presents our approach to overcome them partially. First, we provide some basic rules to model the propagation of deterministic bit-wise linear trails through `XOR` and `Branching` operations.

Proposition 4.6 ($\texttt{XOR}_L$) *Suppose $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is such that $f(x_0, x_1, \ldots, x_{n-1}) = y$ where $y = x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$. The valid deterministic linear trails satisfy*

$$\texttt{XOR}_L\left(\texttt{x}_0, \texttt{x}_1, \ldots, \texttt{x}_{n-1}, \texttt{y}\right) \coloneqq \bigwedge_{i=0}^{n-1} \left(\texttt{x}_i = \texttt{y}\right)$$

where $\texttt{x}_i$, and $\texttt{y}$ are integer variables with the domain $\{-1, 0, 1\}$, for all $0 \leqslant i \leqslant (n-1)$, representing the activeness pattern of linear mask in x_i and y, respectively.

For non-deterministic linear propagations, the propagation rule for branching is the same as the rule for non-deterministic differential propagations through `XOR`. However, we cannot directly apply this duality between an `XOR`'s differential behavior and a branching point's linear behavior when dealing with deterministic propagations. Consider a branching point $f(x) = (y_0, y_1)$, where $y_0 = y_1 = x$. Let $\texttt{x}$, $\texttt{y}_0$, and $\texttt{y}_1$ be integer variables with a domain of $\{-1, 0, 1\}$ representing the activeness pattern of linear masks for x, y_0, and y_1, respectively. Suppose $\texttt{x} = 1$ with certainty. Then, the linear mask of (y_0, y_1) can take either $(1, 0)$ or $(0, 1)$. Therefore, $\texttt{y}_0 = \texttt{y}_1 = -1$. The same is true if $\texttt{x} = 0$. Additionally, if $\texttt{x} = -1$, then $\texttt{y}_0 = \texttt{y}_1 = -1$. This example demonstrates that if we limit ourselves to using a 3-digit encoding (i.e., $\{-1, 0, 1\}$) for modeling the propagation of deterministic linear trails through a branching point, we quickly lose information, and the entire state becomes "-1" (unknown) very quickly. However, suppose that, due to the location of the branching point within the round function (e.g., Feistel structure), the activeness pattern of linear masks at x and y_0 can be derived based on information from the previous round. In that case, we can utilize Proposition 4.7 to model the propagation of deterministic linear propagations through the branching point.

Proposition 4.7 ($\texttt{Branching}_L$) *Suppose $f : \mathbb{F}_2 \to \mathbb{F}_2^n$ is such that $f(x) = (y_0, y_1, \ldots, y_{n-1})$ where $y_0 = y_1 = \ldots = y_{n-1}$. Also assume that the linear masks of x and $y_0, y_1, \cdots, y_{n-2}$ are determined in advance. Then, the valid propagations for deterministic linear trails through the branching point satisfy*

$$\texttt{Branch}_L\left(\texttt{x}, \texttt{y}_0, \ldots, \texttt{y}_{n-1}\right) \coloneqq \begin{cases} \texttt{if} & \left(\bigvee_{i=0}^{n-2} \left(\texttt{y}_i = -1\right)\right) \vee \left(x = -1\right) \ \texttt{then} \ \texttt{y}_{n-1} = -1 \\ \texttt{else} & \texttt{y}_{n-1} = \texttt{x} + \texttt{y}_0 + \ldots + \texttt{y}_{n-2} \bmod 2 \end{cases}$$

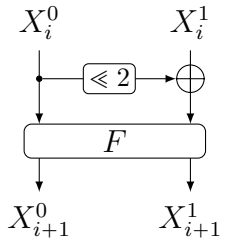
where $\mathtt{x}$ *and* $\mathtt{y}_i$ *are integer variables with the domain* $\{-1, 0, 1\}$ *for all* $0 \leqslant i \leqslant (n-1)$*, representing the activeness pattern of the linear mask in* x *and* y_i*, respectively.*

In addressing the challenge of modeling the propagation of deterministic linear trails through the branching point, our initial focus is on AndRX ciphers, with particular attention to two prominent ones: SIMON [14] and Simeck [19]. We outline our strategy for modeling ZC distinguishers for SIMON, but the same approach applies to Simeck. The idea is to rearrange the state array such that we can model the round function as several consecutive S-boxes along with some branching points, such that the branching points satisfy the requirements of Proposition 4.7. Then we use the rules for propagation of deterministic linear trails through the S-boxes in [13], together with our rule for modeling particular branching points (Proposition 4.7) to model the whole round function.

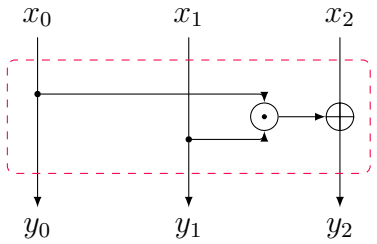
In what follows, we explain the details of our workaround for SIMON. Let X_r^0, and X_r^1 represent the two n-bit input words to the r-th round function of SIMON. The output of the r-th round X_{r+1}^0, X_{r+1}^1 is computed as:

$$\begin{aligned} X_{r+1}^1 &= X_r^0 \\ X_{r+1}^0 &= ((X_r^0 \lll 8) \odot (X_r^0 \lll 1)) \oplus ((X_r^0 \lll 2) \oplus X_r^1) \oplus K_r \end{aligned}$$

Now, we can express the round function of SIMON as illustrated in Figure 4.3. In Figure



(a) One round of SIMON



(b) S_{SIMON}, the core operation of F

Figure 4.3: Round function structure of SIMON, where F is defined by $(y^0, y^1) = F(x^0, x^1) = (x^1 \oplus (x^0 \lll 8) \odot (x^0 \lll 1), x^0)$ and can be expressed in terms of the 3-bit function S_{SIMON}.

4.3, we represent the core operation of the function F: a Toffoli gate (Figure 4.3b). We treat the red dotted box as a 3×3 S-box, referred to as S_{SIMON} hereafter. As illustrated in Figure 4.3b, we represent the input and output of S_{SIMON} by $x = (x_0, x_1, x_2)$, $y = (y_0, y_1, y_2)$, respectively. The algebraic normal form (ANF) of this S-box is as follows:

$$y_0 = x_0, \quad y_1 = x_1, \quad y_2 = (x_0 \odot x_1) \oplus x_2$$

We can represent the function F as a sequence of these S-boxes. Next, we use Proposition 4.8 to encode the propagation of deterministic bit-wise linear propagations through S_{SIMON}.

Proposition 4.8 (Modeling deterministic linear behavior of S_{SIMON}) *Assume that* $\mathtt{x} = (\mathtt{x}_0, \mathtt{x}_1, \mathtt{x}_2)$ *and* $\mathtt{y} = (\mathtt{y}_0, \mathtt{y}_1, \mathtt{y}_2)$ *are integer variables with domain* $\{-1, 0, 1\}$ *to encode the activeness pattern of linear masks at the input and output of* S_{SIMON}*, respectively. The CP*

constraints to describe all valid deterministic bit-wise linear propagations through S_{SIMON} can be derived from its LAT and are summarized as follows:

$$
\begin{cases}
\texttt{if} & (\mathtt{x}_0 = 0 \wedge \mathtt{x}_1 = 0 \wedge \mathtt{x}_2 = 0) & \texttt{then} & (\mathtt{y}_0 = 0 \wedge \mathtt{y}_1 = 0 \wedge \mathtt{y}_2 = 0) \\
\texttt{elseif} & (\mathtt{x}_0 = 0 \wedge \mathtt{x}_1 = 0 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{elseif} & (\mathtt{x}_0 = 0 \wedge \mathtt{x}_1 = 1 \wedge \mathtt{x}_2 = 0) & \texttt{then} & (\mathtt{y}_0 = 0 \wedge \mathtt{y}_1 = 1 \wedge \mathtt{y}_2 = 0) \\
\texttt{elseif} & (\mathtt{x}_0 = 0 \wedge \mathtt{x}_1 = 1 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{elseif} & (\mathtt{x}_0 = 0 \wedge \mathtt{x}_1 = -1 \wedge \mathtt{x}_2 = 0) & \texttt{then} & (\mathtt{y}_0 = 0 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 0) \\
\texttt{elseif} & (\mathtt{x}_0 = 0 \wedge \mathtt{x}_1 = -1 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{elseif} & (\mathtt{x}_0 = 1 \wedge \mathtt{x}_1 = 0 \wedge \mathtt{x}_2 = 0) & \texttt{then} & (\mathtt{y}_0 = 1 \wedge \mathtt{y}_1 = 0 \wedge \mathtt{y}_2 = 0) \\
\texttt{elseif} & (\mathtt{x}_0 = 1 \wedge \mathtt{x}_1 = 0 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{elseif} & (\mathtt{x}_0 = 1 \wedge \mathtt{x}_1 = 1 \wedge \mathtt{x}_2 = 0) & \texttt{then} & (\mathtt{y}_0 = 1 \wedge \mathtt{y}_1 = 1 \wedge \mathtt{y}_2 = 0) \\
\texttt{elseif} & (\mathtt{x}_0 = 1 \wedge \mathtt{x}_1 = 1 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{elseif} & (\mathtt{x}_0 = 1 \wedge \mathtt{x}_1 = -1 \wedge \mathtt{x}_2 = 0) & \texttt{then} & (\mathtt{y}_0 = 1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 0) \\
\texttt{elseif} & (\mathtt{x}_0 = 1 \wedge \mathtt{x}_1 = -1 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{elseif} & (\mathtt{x}_0 = -1 \wedge \mathtt{x}_1 = 0 \wedge \mathtt{x}_2 = 0) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = 0 \wedge \mathtt{y}_2 = 0) \\
\texttt{elseif} & (\mathtt{x}_0 = -1 \wedge \mathtt{x}_1 = 0 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{elseif} & (\mathtt{x}_0 = -1 \wedge \mathtt{x}_1 = 0 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{elseif} & (\mathtt{x}_0 = -1 \wedge \mathtt{x}_1 = 1 \wedge \mathtt{x}_2 = 0) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = 1 \wedge \mathtt{y}_2 = 0) \\
\texttt{elseif} & (\mathtt{x}_0 = -1 \wedge \mathtt{x}_1 = 1 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{elseif} & (\mathtt{x}_0 = -1 \wedge \mathtt{x}_1 = -1 \wedge \mathtt{x}_2 = 0) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 0) \\
\texttt{elseif} & (\mathtt{x}_0 = -1 \wedge \mathtt{x}_1 = -1 \wedge \mathtt{x}_2 = 1) & \texttt{then} & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = 1) \\
\texttt{else} & & & (\mathtt{y}_0 = -1 \wedge \mathtt{y}_1 = -1 \wedge \mathtt{y}_2 = -1)
\end{cases}
$$

Let $\mathtt{x}^{(r)} = (\mathtt{x}_0^r, \mathtt{x}_1^r)$ and $\mathtt{x}^{(r+1)} = (\mathtt{x}_0^{r+1}, \mathtt{x}_1^{r+1})$ denotes the activeness pattern of deterministic linear masks at the input and output of the r-th round of SIMON, respectively. We also assume that $\mathtt{y}_0^r$ and $\mathtt{z}_0^r$ represent the activeness pattern for the linear masks of the two branches of $\mathtt{x}_0^r$ ($\mathtt{z}_0^r$ is one of the branches of $\mathtt{x}_0^r$ which further proceeds into the function F). Then, as depicted in Figure 4.3a, it is evident that this branch fulfills the conditions outlined in Proposition 4.7. Specifically, $\mathtt{x}_0^r$ is derived from the preceding round, while $\mathtt{y}_0^r$ is directly derived from $\mathtt{x}_1^r$, also from the preceding round. As a result, according to Proposition 4.7, we can use the following rule to model this branching point:

$$
\texttt{Branch}_L\left(\mathtt{x}_0^r[i], \mathtt{y}_0^r[i], \mathtt{z}_0^r[i]\right) := \begin{cases} \texttt{if} & (\mathtt{x}_0^r[i] = -1 \ \vee \ \mathtt{y}_0^r[i] = -1) \ \texttt{then} \ \mathtt{z}_0^r[i] = -1 \\ \texttt{else} & \mathtt{z}_0^r[i] = \mathtt{x}_0^r[i] + \ \mathtt{y}_0^r[i] \bmod 2, \end{cases}
$$

where $\mathtt{x}_0^r[i]$, $\mathtt{y}_0^r[i]$, and $\mathtt{z}_0^r[i]$ are integer variables with domain $\{-1, 0, 1\}$, for all $0 \leqslant i \leqslant (n-1)$.

4.2.4 Application of Our Method.

To demonstrate the utility of our improved model for finding ID distinguishers, we applied it to several AndRX (SIMON, Simeck) and ARX (SPECK, LEA, ChaCha, SipHash, Chaskey)

ciphers. Additionally, we successfully utilized our bit-wise modeling to find ZC distinguishers of AndRX ciphers (SIMON, Simeck). While searching for ID and ZC distinguishers, we did not fix any input/output differences. Each bit can take one of three values: 0, 1, or -1 (indicating it can be either 0 or 1). Therefore, having more bits with the value -1 at the input/output of the distinguisher leads to a larger set of distinguishers. Solving one instance of the model returns one solution with several -1s at the input and output, essentially providing a truncated ID/ZC distinguisher. Since bit positions with -1 can take either 0 or 1, the returned solution represents a group or cluster of distinguishers. If there are n input/output bits with -1, we have 2^n distinguishers. Moreover, in the quest for ID (resp. ZC) distinguishers, we incorporate the objective function $\texttt{min}(\sum_{i=0}^{n-1} \texttt{xu}_0[i] + \sum_{i=0}^{n-1} \texttt{xl}_{r_\text{D}}[i])$ aimed at maximizing the count of differentially active bits both at the input and output. The quantity of distinguishers escalates with the increase in the number of undetermined bits at both the input and output of the distinguishers.

While analyzing AndRX and ARX ciphers, we found that both ID and ZC distinguishers of Simeck are based on indirect contradiction. In contrast, for all other ciphers, except Simeck, the best distinguishers are based on direct contradiction. The indirect contradiction technique reveals new differential trails for Simeck but not for other ciphers we analyzed, which we attribute primarily to two factors:

- **Weaker Diffusion:** The diffusion properties of Simeck appear to be weaker compared to SIMON, which leads to more localized propagation of differences. This allows the forward and backward trails to more easily merge and create contradictions, especially when we propagate the merged information with certainty.

- **Round Structure Differences (Bit Rotation):** The round functions of Simeck and SIMON differ significantly in their handling of bit rotations. Simeck's round structure, which incorporates bit rotations, may result in more predictable propagation patterns of the input differences, making it more susceptible to indirect contradictions. In contrast, SIMON's round function, with different manipulations of the state, may reduce the predictability of the difference propagation, thereby preventing such contradictions from appearing.

These factors combined may explain why indirect contradictions arise in Simeck, making Simeck more vulnerable to this cryptanalytic technique.

Figure 4.8, and Figure 4.4 showcase examples of ID (resp. ZC) distinguishers identified using our tool. In both the forward and backward propagations, we denote the undetermined bits (difference or linear mask) as ◩ and ◪, respectively. The active bit (bit difference or linear mask value of 1) is represented as ◩ and ◪ in the forward and backward processes, respectively. According to our modeling, for all $0 \leqslant r \leqslant r_\text{D}$, the CP variables $\texttt{xu}_r$ and $\texttt{xl}_r$ are represented in the upper triangle and lower triangle of the rth state, $L_r||R_r$, respectively. For instance, Figure 4.8 shows the ID distinguishers for 6-round SPECK-96, and Figure 4.9 for 19-round SIMON128. In Figure 4.8, the 42-th bit of R_2 has difference values of 1 and 0 in the forward and backward propagation, respectively, indicating a 0-1 contradiction in 6-round SPECK-96.

Here, Figure 4.4 illustrates the ZC distinguisher of 15-round Simeck, demonstrating the use of indirect contradiction. According to our model, the CP variable $\texttt{xu}_r$ is represented

in the upper triangle of the r-th state, $L_r||R_r$ (on the left side column of Figure 4.4) for all $0 \leqslant r \leqslant r_{\text{M}} = 8$. Similarly, the CP variable $\texttt{xl}_r$ is represented in the lower triangle of the r-th state, $L_r||R_r$ (on the left side column of Figure 4.4) for all $8 = r_{\text{M}} \leqslant r \leqslant r_{\text{D}} = 15$.

Following our model to find an indirect contradiction, the variable $\texttt{mx}_{r_{\text{M}}}$ (represented in both the upper and lower triangle of $L_{\text{M}}||R_{\text{M}}$, on the right side column of Figure 4.4 with $r_{\text{M}} = 8$) is the result of merging of two variables $\texttt{xu}_{r_{\text{M}}}$ (upper triangle of $L_{\text{M}}||R_{\text{M}}$, on the left side column of Figure 4.4 with $r_{\text{M}} = 8$), and $\texttt{xl}_{r_{\text{M}}}$ (lower triangle of $L_{\text{M}}||R_{\text{M}}$, on the left side column of Figure 4.4 with $r_{\text{M}} = 8$). Now, for all $0 \leqslant r \leqslant r_{\text{M}}$, we first propagate $\texttt{mx}_r$ (lower triangle of the state $L_r||R_r$ on the right side column of Figure 4.4) to $\texttt{mx}'_{r-1}$ (lower triangle of the state $L'_{r-1}||R'_{r-1}$ on the right side column of Figure 4.4) backward through one round function. Then, we merge activeness pattern of $\texttt{mx}'_{r-1}$, and $\texttt{xu}_{r-1}$ into $\texttt{mx}_{r-1}$, and check whether there is a contradiction between $\texttt{mx}'_{r-1}$, and $\texttt{xu}_{r-1}$. For example, we can see in Figure 4.4, the value of $\texttt{xu}_2[0]$ (depicted in the upper triangle of the 0-th bit of L_2 in the left column) is 1, while the value of $\texttt{mx}'_2[0]$ (depicted in the lower triangle of the 0-th bit of L'_2 in the right column) is 0. This implies an indirect contradiction occurs in 15-round Simeck-48. Similarly, for all $r_{\text{M}} \leqslant r \leqslant r_{\text{D}}$, we propagate $\texttt{mx}_r$ (upper triangle of the state $L_r||R_r$ on the right side column of Figure 4.4) into $\texttt{mx}'_{r+1}$ (upper triangle of the state $L'_{r+1}||R'_{r+1}$ on the right side column of Figure 4.4), and merge $\texttt{mx}'_{r+1}$ with $\texttt{xl}_{r+1}$ into $\texttt{mx}_{r+1}$ and check if there is a contradiction between $\texttt{mx}_{r+1}$ and $\texttt{xl}_{r+1}$. This type of contradiction (represented in Figure 4.4) cannot be detectable by the previous methods.

Unlike previous tools based on unsatisfiability [94], [95], which require multiple executions by fixing the input and output of the distinguisher in each run, our tool efficiently identifies a group of ID/ZC distinguishers (or truncated distinguishers) in just one execution, without the need to fix input/output differences (as detailed in Tables 3-7). A single execution of our tool terminates within a few seconds (or minutes) for our models based on direct contradiction (or indirect contradiction) when running on a regular laptop.

4.2.5 Comparison of Our Distinguisher Modeling to Prior Methods.

Building on the methodology outlined in [12], [13], our goal with the CP models for distinguishers is to develop satisfiability-based frameworks capable of unifying into a comprehensive COP for identifying complete ID and ZC attacks. Unlike previous methodologies like those discussed in [93]–[95], which rely on unsatisfiability and require fixing input/output differences or linear masks to detect distinguishers, our attack model operates on satisfiability principles. Our method removes the requirement to fix input/output differences or linear masks. Moreover, our bit-wise CP model for distinguishers demonstrates a novel capability by identifying ID and ZC distinguishers based on indirect contradictions, improving on previous ones [12], [13], which only handled direct contradictions. We tested our model on various ciphers. Interestingly, only Simeck yielded longer distinguishers based on indirect contradiction compared to the direct contradiction approach. In other AndRX and ARX applications, the improved distinguishers we found, along with the longest existing ones, could be explained or found through direct contradiction. Still, we believe this doesn't lessen our contribution with indirect contradiction, as our model is overall more complete than its predecessor [12], [13], since the previous models in [12], [13] cannot find the longest existing ID and ZC distinguishers of Simeck as we did. Furthermore, our enhanced bit-wise

model, which utilizes satisfiability for identifying ID/ZC distinguishers, can be extended into a unified COP model to uncover full ID attacks on both SIMON and Simeck. Consequently, our model enables the discovery of improved ID attacks on all versions of Simeck, a feat beyond the capabilities of previous tools [12], [13].

Although our primary objective in modeling distinguishers is not to develop tools for proving the non-existence of ID/ZC distinguishers, it is worth discussing whether our model could also fulfill this purpose. We cannot claim that our tool can capture the longest possible ID/ZC distinguisher or our tools can be used to prove the non-existence of ID/ZC distinguishers because we rely on the assumptions of round independence and subkey independence. However, this limitation is not exclusive to our approach; it applies to all existing tools for finding ID/ZC distinguishers. Even the tool developed in [94], which captures complex contradictions, requires checking every possible input/output combination for non-existence, which is impractical. Developing a tool for proving non-existence remains an exciting future direction. However, our current focus is on creating a satisfiability-based model for ID (resp. ZC) distinguishers that can later be adapted for key recovery strategies. Nevertheless, our tools have proven effective in discovering the best existing distinguishers to date, generating numerous new trails, and revisiting older ones, all accomplished within minutes on a standard laptop. These applications include SIMON, Simeck, SPECK, ChaCha, LEA, SipHash, and Chaskey.

In our model, we account for the fact that the AND gates in one round share certain input bits. Concerning cross-round dependencies, akin to previous tools for searching for ID/ZC, we assume that consecutive non-linear operations (rounds) are statistically independent. Consequently, like previous tools, any ID/ZC distinguisher we identify remains valid. However, it is worth noting that, similar to previous tools, we may overlook some ID/ZC distinguishers that are detectable only by considering cross-round dependencies. Nevertheless, it is important to acknowledge that addressing cross-round dependencies has been an open problem in the context of ID/ZC distinguishers so far. While we are aware of recent works that consider cross-round dependencies for differential characteristics (e.g., [118]), these works primarily address dependency issues for differential characteristics (single trail) and are not applicable to differentials (i.e., differential hulls) and ID/ZC distinguishers. Therefore, we have chosen to keep our model simple, efficient, and based on satisfiability to facilitate its extension for key recovery, which has been the main motivation of this work and the underlying methods [12], [13]. Regarding key recovery, our approach aligns with previous works, wherein we utilize deterministic properties and refrain from exploiting any properties that may conflict with the fact that certain non-linear operations may have dependencies.

In summary, our new bit-wise models bridge the gap in developing satisfiability-based models for distinguishers applicable to ARX and AndRX ciphers. Along with that, they upgrade the distinguisher models by handling the detection of both direct and indirect contradictions. In Section 4.3, we demonstrate the extension of our CP models for distinguishers into a comprehensive framework for identifying complete ID attacks.

4.3 Key-Recovery Modeling for Impossible Differentials

Here, we introduce an approach that expands the distinguisher model into a unified approach for uncovering comprehensive ID attacks, encompassing key recovery for AndRX ciphers. Our framework takes four integer parameters (r_{B}, r_{D}, r_{M}, r_{F}) that represent the lengths of specific parts in Figure 2.8, where r_{M} specifies the merging point throughout the distinguisher part as explained in Subsection 4.2.2 and generates an optimum ID attack for $r = r_{\mathrm{B}} + r_{\mathrm{D}} + r_{\mathrm{F}}$ rounds. In pursuing the full ID attack, our main goal is to minimize the overall time complexity while adhering to specified limits on memory and data complexity.

After reviewing the complexity formulas detailed in Equation 4.6, the parameters c_{B}, c_{F}, $|\Delta_{\mathrm{B}}|$, $|\Delta_{\mathrm{F}}|$, and $|k_{\mathrm{B}} \cup k_{\mathrm{F}}|$ are pivotal factors that directly influence the comprehensive complexity of the ID attack.

In identifying $(c_{\mathrm{B}}, \Delta_{\mathrm{B}})$, it is crucial to analyze how truncated differential trails propagate through E_{B}, accounting for the probabilities of all propagations from a `truncated difference` to a `fixed difference`. Detecting k_{B} involves identifying the state bits that undergo differential or data value changes during partial encryption within E_{B}. Similarly, during partial decryption using E_{F}^{-1}, we identify c_{F}, $|\Delta_{\mathrm{F}}|$, and k_{F}. Additionally, to ascertain the size of $k_{\mathrm{B}} \cup k_{\mathrm{F}}$, techniques such as the *equivalent sub-key technique* for Feistel ciphers or the *key bridging technique* can be employed.

4.3.1 Brief Overview of the COP model

Our bit-wise key recovery model consists of 4 sub-models as follows:

- **Modeling the distinguisher.** We model the distinguisher part according to the method explained in Section 4.2.
- **Modeling the difference propagation through E_{B} and E_{F}.** In this part, we model truncated differential propagation $\Delta_{\mathrm{B}} \xleftarrow{E_{\mathrm{B}}^{-1}} \Delta_{\mathrm{U}}$ and truncated differential propagation $\Delta_{\mathrm{L}} \xrightarrow{E_{\mathrm{F}}} \Delta_{\mathrm{F}}$. There is a similarity between this modeling and the modeling of distinguishers: Both use integer variables of domain $\{-1, 0, 1\}$ for each state bit. We model the deterministic propagation in both forward and backward trails using straightforward rules that govern the transmission through fundamental operations like `XOR`, `AND`, `Branch`, and others. We also employ novel binary variables and constraints to model the quantities of filters c_{B} and c_{F} capturing the probabilities of $\Delta_{\mathrm{B}} \xrightarrow{E_{\mathrm{B}}} \Delta_{\mathrm{U}}$ and $\Delta_{\mathrm{L}} \xleftarrow{E_{\mathrm{F}}^{-1}} \Delta_{\mathrm{L}}$.
- **Modeling guess-and-determine.** Here, we analyze the dependency relationships between E_{B} and E_{F} to pinpoint the state bits whose differences or data values are crucial for validating the transformations from Δ_{B} to Δ_{U} through E_{B}, and from Δ_{F} to Δ_{L} through E_{F}^{-1}. We can find the key bits involved in the key recovery attack by utilizing this information.
- **Modeling the complexity formula.** We address the complexity formulas detailed in Section 2.2.4, culminating in the objective of minimizing `T`, representing the total time complexity in our CP model. This involves minimizing the maximal term within the time complexity framework.

The variables within our model are either integers or binary, adhering to the domain $\{-1, 0, 1\}$, whereas the variables related to the complexity formula are real numbers. Our tool exclusively relies on four integer inputs, delineating the lengths of E_{B} (r_{B}), E_{D} (r_{D}), E_{U} (r_{M}), and E_{F} (r_{F}). Users have the flexibility to experiment with various length configurations for these components to discover an optimal ID attack strategy. Additionally, the model's objective function can be adapted to minimize data or memory complexities under constraints such as time or other parameters.

4.3.2 Detailed Description of Bit-Wise Key Recovery Model and Application to SIMON

We provide an in-depth explanation of the bit-wise model used to conduct a comprehensive ID attack on AndRX ciphers. As an example, we formulate the COP model to detect a complete ID attack on SIMON. The model focuses on four integer parameters: r_{B}, r_{D}, r_{M}, and r_{F}. This approach targets $r = r_{\mathrm{B}} + r_{\mathrm{D}} + r_{\mathrm{F}}$ rounds of SIMON, with r_{D} representing the distinguisher length, and r_{M} is the round where one can find a contradiction (in case of direct contradiction), or one can find some indirect contradiction where r_{M} is the round where the first time merging operation happens ($r_{\mathrm{M}} = 8$ in Figure 4.4). To perform key recovery (as shown in Figure 2.8), we must extend the distinguisher by a few rounds at both ends. r_{B} and r_{F} are the lengths of extended backward and forward parts, respectively.

Modeling the Distinguisher

Initially, our aim is to represent the propagation of differences through the round function of SIMON. For the detailed structure of the round function SIMON, please refer to Subsection 2.4.1. Here, we define $\mathtt{xu}_r^0$ and $\mathtt{xu}_r^1$ to be the CP variables corresponding to the two n-bit input words to the r-th round function of SIMON, where the block size of SIMON is $2n$. In more detail, for all $0 \leqslant i \leqslant (n-1)$, $\mathtt{xu}_r^0[i]$, and $\mathtt{xu}_r^1[i]$ are integer variables with domain $\{-1, 0, 1\}$. Within the data path of SIMON, operations such as AND and XOR alter the state's difference pattern, while rotations (ROT) shift the position of this difference pattern during the propagation of deterministic differences. We described the rules for deterministic differential propagation through these basic operations in Subsection 4.2.1. Then, we construct the model $\mathtt{CSP}_{\mathrm{U}}$ for the upper part as described in Algorithm 3. Similarly, we can construct $\mathtt{CSP}_{\mathrm{L}}$. Along with this, we also build the $\mathtt{CSP}_{\mathrm{M}}$, $\mathtt{CSP}_{\mathrm{B}}$, $\mathtt{CSP}_{\mathrm{F}}$, and $\mathtt{CSP}_{\mathrm{C}}$ according to Equation 4.2, Equation 4.3, Equation 4.4, and Equation 4.5, respectively. The combined CSP model is $\mathtt{CSP}_{\mathrm{D}} := \mathtt{CSP}_{\mathrm{U}} \wedge \mathtt{CSP}_{\mathrm{L}} \wedge \mathtt{CSP}_{\mathrm{M}} \wedge \mathtt{CSP}_{\mathrm{F}} \wedge \mathtt{CSP}_{\mathrm{B}} \wedge \mathtt{CSP}_{\mathrm{C}}$. Therefore, any feasible solution of $\mathtt{CSP}_{\mathrm{D}}$ corresponds to an ID distinguisher for SIMON with block size $2n$.

Modeling the Difference Propagation through E_{B} and E_{F}

For the deterministic difference propagations $\Delta_{\mathrm{B}} \xleftarrow{E_{\mathrm{B}}^{-1}} \Delta_{\mathrm{U}}$, and $\Delta_{\mathrm{L}} \xrightarrow{E_{\mathrm{F}}} \Delta_{\mathrm{F}}$, we assign integer variables with values in $-1, 0, 1$ to each state bit. These variables denote whether the difference value is 0, 1, or undetermined. Hence, utilizing the propagation rules for deterministic differential propagation through basic operations (AND, XOR, ROT), we can determine

Algorithm 3 $\texttt{CSP}_{\text{U}}$ model of difference propagation through E_{D} for SIMON

Input: The integer number r_{D}
Output: $\texttt{CSP}_{\text{U}}$
Declare an empty CSP model $\mathcal{M}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{xu}_r^0[i] \in \{-1,0,1\},\ \texttt{xu}_r^1[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{D}},\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{yu}_r^0[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{D}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{zu}_r^0[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{D}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{wu}_r^0[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{D}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{pu}_r^0[i] \in \{-1,0,1\},\ \texttt{qu}_r^0[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{D}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
for $r = 0, \ldots, r_{\text{D}}-1$ **do**
    $\mathcal{M}.\texttt{con} \leftarrow \texttt{yu}_r^0 = \texttt{xu}_r^0 \lll 8$
    $\mathcal{M}.\texttt{con} \leftarrow \texttt{zu}_r^0 = \texttt{xu}_r^0 \lll 1$
    $\mathcal{M}.\texttt{con} \leftarrow \texttt{wu}_r^0 = \texttt{xu}_r^0 \lll 2$
for $r = 0, \ldots, r_{\text{D}}-1,\ i = 0, \ldots (n-1)$ **do**
    $\mathcal{M}.\texttt{con} \leftarrow \texttt{AND}(\texttt{yu}_r^0[i], \texttt{zu}_r^0[i], \texttt{pu}_r^0[i])$
for $r = 0, \ldots, r_{\text{D}}-1,\ i = 0, \ldots (n-1)$ **do**
    $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}(\texttt{pu}_r^0[i], \texttt{wu}_r^0[i], \texttt{qu}_r^0[i])$
for $r = 0, \ldots, r_{\text{D}}-1,\ i = 0, \ldots (n-1)$ **do**
    $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}(\texttt{qu}_r^0[i], \texttt{xu}_r^1[i], \texttt{xu}_{r+1}^0[i])$
for $r = 0, \ldots, r_{\text{D}}-1,\ i = 0, \ldots (n-1)$ **do**
    $\mathcal{M}.\texttt{con} \leftarrow (\texttt{xu}_r^0[i] = \texttt{xu}_{r+1}^1[i])$
return $\mathcal{M}$

the deterministic difference backward propagation starting from Δ_{U}, and the deterministic difference forward propagation starting from Δ_{L}.

To model the probability of difference propagations $\Delta_{\text{B}} \xrightarrow{E_{\text{B}}} \Delta_{\text{U}}$ and $\Delta_{\text{L}} \xleftarrow{E_{\text{F}}^{-1}} \Delta_{\text{F}}$, we should identify probabilistic transitions. The probabilistic transitions in our modeling are those `truncated difference` $\rightarrow$ `fixed difference` throughout `XOR` operations. For example, if at least one of the two input differences of the `XOR` operation is truncated (unknown) but its output difference is fixed, then we have a `truncated difference` $\rightarrow$ `fixed difference` transition. We call such a transition probabilistic. Given that the primary source of probabilistic transitions through $\Delta_{\text{B}} \rightarrow \Delta_{\text{U}}$ and $\Delta_{\text{L}} \leftarrow \Delta_{\text{F}}$ in our modeling are probabilistic transitions through the `XOR` operations, we provide a basic rule to identify the probabilistic transitions for `XOR`. Let $z = x \oplus y$, where $x, y, z \in \mathbb{F}_2$. Additionally, $\texttt{dx}, \texttt{dy}, \texttt{dz} \in \{-1, 0, 1\}$ are integer variables to encode the difference at the input and output of the `XOR` operation. We define a binary variable `cb` to indicate whether there is a probabilistic transition over the corresponding `XOR` operation. Then, we use the following constraint to determine the value of `cb`:

$$\texttt{XOR}_{dp}(\texttt{dx}, \texttt{dy}, \texttt{dz}, \texttt{cb}) \coloneqq \texttt{if}\ (\texttt{dz} \geqslant 0 \wedge (\texttt{dx} = -1 \vee \texttt{dy} = -1))\ \texttt{then}\ \texttt{cb} = 1\ \texttt{else}\ \texttt{cb} = 0$$

We use this constraint for each `XOR` throughout E_{B} and E_{F}. For this purpose, we define a binary variable $\texttt{cb}_r^j[i]$ ($\texttt{cf}_r^j[i]$) for each `XOR` operation in the r-th round of E_{B} (E_{F}), where $0 \leqslant i \leqslant (n-1)$, and $1 \leqslant j \leqslant t$ such that t is the total number of `XOR` operations in one round function. For SIMON, there are two `XOR` operations in one round. Algorithm 4 outlines our approach to difference propagation across E_{B}. Likewise, a corresponding model can be developed for difference propagation across E_{F}. Finally, we combine $\texttt{CSP}_{\text{B}}^{dp}$ and $\texttt{CSP}_{\text{L}}^{dp}$ into $\texttt{CSP}_{\text{DP}} \coloneqq \texttt{CSP}_{\text{B}}^{dp} \wedge \texttt{CSP}_{\text{F}}^{dp}$ to represent the propagation of differences through the outer components.

Algorithm 4 $\texttt{CSP}_{\text{B}}^{dp}$ model of difference propagation through E_{B} for SIMON

Input: $\texttt{CSP}_{\text{U}}.\texttt{var}$, and the integer number r_{B}
Output: $\texttt{CSP}_{\text{B}}^{dp}$
Declare an empty CSP model $\mathcal{M}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{dxu}_r^0[i] \in \{-1,0,1\},\ \texttt{dxu}_r^1[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}},\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{dyu}_r^0[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{dzu}_r^0[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{dwu}_r^0[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{dpu}_r^0[i] \in \{-1,0,1\},\ \texttt{dqu}_r^0[i] \in \{-1,0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{cb}_r^1[i] \in \{0,1\},\ \texttt{cb}_r^2[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
for $i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{dxu}_{r_{\text{B}}}^0[i] = \texttt{xu}_0^0[i])$
for $i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{dxu}_{r_{\text{B}}}^1[i] = \texttt{xu}_0^1[i])$
for $r = 0, \ldots, r_{\text{B}}-1$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{dyu}_r^0 = \texttt{dxu}_{r+1}^1 \lll 8$
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{dzu}_r^0 = \texttt{dxu}_{r+1}^1 \lll 1$
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{dwu}_r^0 = \texttt{dxu}_{r+1}^1 \lll 2$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{AND}(\texttt{dyu}_r^0[i], \texttt{dzu}_r^0[i], \texttt{dpu}_r^0[i])$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}(\texttt{dpu}_r^0[i], \texttt{dwu}_r^0[i], \texttt{dqu}_r^0[i])$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}(\texttt{dqu}_r^0[i], \texttt{dxu}_{r+1}^0[i], \texttt{dxu}_r^1[i])$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{dxu}_r^0[i] = \texttt{dxu}_{r+1}^1[i])$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}_{dp}(\texttt{dpu}_r^0[i], \texttt{dwu}_r^0[i], \texttt{dqu}_r^0[i], \texttt{cb}_r^1[i])$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}_{dp}(\texttt{dqu}_r^0[i], \texttt{dxu}_r^1[i], \texttt{dxu}_{r+1}^0)[i], \texttt{cb}_r^2[i])$
return $\mathcal{M}$

Modeling Guess-and-Determine

Identifying the state bits crucial for evaluating the bit conditions in $\Delta_{\text{B}} \xrightarrow{E_{\text{B}}} \Delta_{\text{U}}$ and $\Delta_{\text{L}} \xleftarrow{E_{\text{F}}^{-1}} \Delta_{\text{F}}$, considering their differences or values, or both. We begin by identifying the state bits essential for which their differences are required. The state bit's difference value is essential when it influences a bit condition or equivalently probabilistic transition. In our model, we propagate Δ_{U} to Δ_{B} (resp. Δ_{L} to Δ_{F}) with probability one, whereas for key recovery, we process the data in the opposite direction. Therefore, considering the Feistel structure of SIMON and Simeck, when processing data from Δ_{B} to Δ_{U} and (resp. from Δ_{F} to Δ_{L}), probabilistic transitions (`truncated difference` → `fixed difference`) only occur through XOR operations, while propagations through AND operations are deterministic.

For example, there are two bits x, and y such that $z = x \oplus y$. `dx`, `dy`, and `dz` be integer variables with domain $\{-1, 0, 1\}$ indicating the deterministic difference pattern of x, y, and z, and `cb` be the binary variable depicting whether there is a probabilistic transition through `XOR`. Moreover, `kdx`, `kdy`, and `kdz` are binary variables indicating whether the values of the differences of x, y, and z are needed or not. Our goal is to predict the values of `kdx` and `kdy` given the values of `dx`, `dy`, `kdz`, and `cb`. We analyze all possible cases:

- When $\texttt{kdz} = \texttt{cb} = 0$, it means that the value of the difference in the z position is not needed, and there is no probabilistic transition through the `XOR` operation. Consequently, the differences in x and y positions are also not needed, implying that $\texttt{kdx} = \texttt{kdy} = 0$.

- If either `kdz` or `cb` equals 1, then the values of `kdx` and `kdy` depend on the values of `dx` and `dy`. For instance, if `dx` is 0 or 1, this indicates that the value of the difference at position x is already known, which means $\texttt{kdx} = 0$. Conversely, if $\texttt{dx} = -1$, it signifies that the value of the difference at position x is unknown, leading to $\texttt{kdx} = 1$. A similar analysis applies to `dy`.

Therefore, based on the aforementioned concept, we state the following proposition.

Proposition 4.9 ($\texttt{XOR}_1^{gd}$) *Suppose $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ is such that $z = f(x, y) = x \oplus y$. Let* `dx`*,* `dy`*, and* `dz` *be integer variables with domain $\{-1, 0, 1\}$ indicating the deterministic difference pattern of x, y, and z, and* `cb` *be the binary variable depicting whether there is a probabilistic transition through* `XOR`*. Then, the valid propagations for detecting bit cells whose values of the differences are needed through the* `XOR` *operation should satisfy the following constraints:*

$$
\begin{aligned}
&\texttt{XOR}_1^{gd}(\texttt{dx}, \texttt{dy}, \texttt{kdz}, \texttt{cb}, \texttt{kdx}, \texttt{kdy}) := \\
&\begin{cases}
\texttt{if} & (\texttt{kdz} = 0 \wedge \texttt{cb} = 0) & \texttt{then } (\texttt{kdx} = 0 \wedge \texttt{kdy} = 0) \\
\texttt{elseif} & ((\texttt{kdz} + \texttt{cb}) \geqslant 1 \wedge \texttt{dx} = -1 \wedge \texttt{dy} \geqslant 0) & \texttt{then } (\texttt{kdx} = 1 \wedge \texttt{kdy} = 0) \\
\texttt{elseif} & ((\texttt{kdz} + \texttt{cb}) \geqslant 1 \wedge \texttt{dx} \geqslant 0 \wedge \texttt{dy} = -1) & \texttt{then } (\texttt{kdx} = 0 \wedge \texttt{kdy} = 1) \\
\texttt{elseif} & ((\texttt{kdz} + \texttt{cb}) \geqslant 1 \wedge \texttt{dx} = -1 \wedge \texttt{dy} = -1) & \texttt{then } (\texttt{kdx} = 1 \wedge \texttt{kdy} = 1) \\
\texttt{else} & & (\texttt{kdx} = 0 \wedge \texttt{kdy} = 0)
\end{cases}
\end{aligned}
$$

where `kdx`*,* `kdy`*, and* `kdz` *are binary variables indicating whether the values of the differences of x, y, and z are needed or not.*

Algorithm 5 $\texttt{CSP}_{\text{B}}^{gd}$ model for guess-and-determine through E_{B} for SIMON

Input: $\texttt{CSP}_{\text{U}}.\texttt{var}$, $\texttt{CSP}_{\text{B}}^{dp}$, and the integer number r_{B}
Output: $\texttt{CSP}_{\text{B}}^{gd}$
Declare an empty CSP model $\mathcal{M}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kdxu}_r^0[i] \in \{0,1\},\ \texttt{kdxu}_r^1[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}},\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kdyu}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kdzu}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kdwu}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kdpu}_r^0[i] \in \{0,1\},\ \texttt{kdqu}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kdyu1}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kdzu1}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kdwu1}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kxu}_r^0[i] \in \{0,1\},\ \texttt{kxu}_r^1[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}},\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kyu}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kzu}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kwu}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kpu}_r^0[i] \in \{0,1\},\ \texttt{kqu}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kyu1}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kzu1}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{kwu1}_r^0[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
$\mathcal{M}.\texttt{var} \leftarrow \{\texttt{ikb}_r[i] \in \{0,1\} :\ 0 \leqslant r \leqslant r_{\text{B}}-1,\ 0 \leqslant i \leqslant (n-1)\}$
for $i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{if}\ \texttt{dxu}_{r_{\text{B}}}^0[i] \geqslant 0\ \texttt{then}\ \texttt{kdxu}_{r_{\text{B}}}^0[i] = 0\ \texttt{else}\ \texttt{true})$
for $i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{if}\ \texttt{dxu}_{r_{\text{B}}}^1[i] \geqslant 0\ \texttt{then}\ \texttt{kdxu}_{r_{\text{B}}}^1[i]\ \texttt{else}\ \texttt{true})$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}_1^{gd}(\texttt{dxu}_r^1[i], \texttt{dqu}_r^0[i], \texttt{kdxu}_{r+1}^0[i], \texttt{cb}_r^2[i], \texttt{kdxu}_r^1[i], \texttt{kdqu}_r^0[i])$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}_1^{gd}(\texttt{dqu}_r^0[i], \texttt{dwu}_r^0[i], \texttt{kdqu}_r^0[i], \texttt{cb}_r^1[i], \texttt{kdpu}_r^0[i], \texttt{kdwu}_r^0[i])$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{AND}_1^{gd}(\texttt{dyu}_r^0[i], \texttt{dzu}_r^0[i], \texttt{kdpu}_r^0[i], \texttt{kdyu}_r^0[i], \texttt{kdzu}_r^0[i])$
for $r = 0, \ldots, r_{\text{B}}-1$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{kdyu}_r^0 = \texttt{kdyu1}_r^0 \gg 8$
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{kdzu}_r^0 = \texttt{kdzu1}_r^0 \gg 1$
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{kdwu}_r^0 = \texttt{kdwu1}_r^0 \gg 2$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{if}\ \texttt{kdyu1}_r^0[i] = \texttt{kdzu1}_r^0[i] = \texttt{kdwu1}_r^0[i] = \texttt{kdxu}_{r+1}^1[i] = 0\ \texttt{then}\ \texttt{kdxu}_r^0[i] = 0\ \texttt{else}\ \texttt{kdxu}_r^0[i] = 1)$
for $i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{kxu}_{r_{\text{B}}}^0[i] = 0)$
for $i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{kxu}_{r_{\text{B}}}^1[i] = 0)$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}_2^{gd}(\texttt{kxu}_{r+1}^0[i], \texttt{kxu}_r^1[i], \texttt{kqu}_r^0[i])$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{XOR}_2^{gd}(\texttt{kqu}_r^0[i], \texttt{kpu}_r^0[i], \texttt{kwu}_r^0[i])$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{AND}_2^{gd}(\texttt{kdpu}_r^0[i], \texttt{kpu}_r^0[i], \texttt{dyu}_r^0[i], \texttt{dzu}_r^0[i], \texttt{kyu}_r^0[i], \texttt{kyu}_r^0[i])$
for $r = 0, \ldots, r_{\text{B}}-1$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{kyu}_r^0 = \texttt{kyu1}_r^0 \gg 8$
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{kzu}_r^0 = \texttt{kzu1}_r^0 \gg 1$
  $\mathcal{M}.\texttt{con} \leftarrow \texttt{kwu}_r^0 = \texttt{kwu1}_r^0 \gg 2$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{if}\ \texttt{kyu1}_r^0[i] = \texttt{kzu1}_r^0[i] = \texttt{kwu1}_r^0[i] = \texttt{kxu}_{r+1}^1[i] = 0\ \texttt{then}\ \texttt{kxu}_r^0[i] = 0\ \texttt{else}\ \texttt{kxu}_r^0[i] = 1)$
for $r = 0, \ldots, r_{\text{B}}-1,\ i = 0, \ldots (n-1)$ **do**
  $\mathcal{M}.\texttt{con} \leftarrow (\texttt{if}\ \texttt{kxu}_{r+1}^0[i] = 1\ \texttt{then}\ \texttt{ikb}_r[i] = 1\ \texttt{else}\ \texttt{ikb}_r[i] = 0)$
return $\mathcal{M}$

Additionally, we can construct the necessary constraints to propagate the state bits with the required difference value through the AND operation similarly. In this context, we analyze the possible cases in the following way:

- If $\texttt{kdz} = 0$, it means that the value of the difference in the z position is not needed. Therefore, the differences in the x, and y position also not needed, implying that $\texttt{kdx} = \texttt{kdy} = 0$.
- If $\texttt{kdz} = 1$, then the values of kdx and kdy depend on the values of dx, and dy.

Hence, we state the following proposition:

Proposition 4.10 ($\texttt{AND}_1^{gd}$) *Suppose $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ is such that $z = f(x, y) = x \cdot y$. Let* dx, dy, *and* dz *be integer variables with domain $\{-1, 0, 1\}$ indicating the deterministic difference pattern of x, y, and z, and* kdx, kdy, *and* kdz *binary variables indicating whether the values of the differences of x, y, and z are needed or not. Then, the valid propagations for detecting state bits whose values of the differences are needed through the* ***AND*** *operation satisfy the following constraints:*

$$\texttt{AND}_1^{gd}(\texttt{dx}, \texttt{dy}, \texttt{kdz}, \texttt{kdx}, \texttt{kdy}) :=$$

$$\begin{cases} \texttt{if} & (\texttt{kdz} = 0) & \texttt{then } (\texttt{kdx} = 0 \wedge \texttt{kdy} = 0) \\ \texttt{elseif} & (\texttt{dx} = -1 \wedge \texttt{dy} \geqslant 0 \wedge \texttt{kdz} = 1) & \texttt{then } (\texttt{kdx} = 1 \wedge \texttt{kdy} = 0) \\ \texttt{elseif} & (\texttt{dx} \geqslant 0 \wedge \texttt{dy} = -1 \wedge \texttt{kdz} = 1) & \texttt{then } (\texttt{kdx} = 0 \wedge \texttt{kdy} = 1) \\ \texttt{elseif} & (\texttt{dx} = -1 \wedge \texttt{dy} = -1 \wedge \texttt{kdz} = 1) & \texttt{then } (\texttt{kdx} = 1 \wedge \texttt{kdy} = 1) \\ \texttt{else} & & (\texttt{kdx} = 0 \wedge \texttt{kdy} = 0) \end{cases}$$

Therefore, we define binary variables for each state bit through E_{B} and E_{F} to indicate whether the difference value of each state bit over E_{B} and E_{F} is needed, and using Proposition 4.9 and Proposition 4.10, we encode the propagation of state bits whose difference value is needed. Additionally, we introduce a novel constraint that connects the start of E_{U} with the conclusion of E_{B}, and the conclusion of E_{L} with the commencement of E_{F}. When considering the determination of data values, the nonlinear operation AND becomes relevant. We describe this determination over the functions in E_{B}, although a similar model can be applied to E_{F}. Let's suppose $z = x \cdot y$, where $x, y, z \in \mathbb{F}_2$. Assuming δx, δy, and δz represent the values of the differences in x, y, and z, respectively, we have: $\delta z = (x \cdot y) \oplus ((x \oplus \delta x) \cdot (y \oplus \delta y))$. Now, let's assume the values of the differences δx and δy are known, and we aim to determine the value of δz, for instance, to check a filter. Additionally, assume that we do not need to know the value of z. Consequently, whether we need the value of x or y to determine δz depends on δx and δy. For instance, if $\delta y = 0$, we do not need to know the value of x to derive δz. Similarly, if $\delta x = 0$, we do not need to know the value of y to determine δz. Thus, it is crucial to consider the value of differences when modeling the AND operation in guess-and-determine. Proposition 4.11 outlines how to model the AND operation in guess-and-determine.

Proposition 4.11 ($\texttt{AND}_2^{gd}$) *Suppose $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ is such that $z = f(x, y) = x \cdot y$. Let* dx, dy, *and* dz *be integer variables with domain $\{-1, 0, 1\}$ indicating the deterministic difference*

pattern of x, y, and z, and $\texttt{kdz}$ *binary variable indicating whether the values of the differences of z are needed. Then, the valid propagations for detecting state bits whose values of the differences are needed through the* ***AND*** *operation satisfy the following constraints:*

$$\texttt{AND}_2^{gd}(\texttt{kdz}, \texttt{kz}, \texttt{dx}, \texttt{dy}, \texttt{kx}, \texttt{ky}) :=$$

$$\begin{cases} \texttt{if} & (\texttt{kdz} = 0 \wedge \texttt{kz} = 0) & \texttt{then } (\texttt{kx} = 0 \wedge \texttt{ky} = 0) \\ \texttt{elseif} & (\texttt{kdz} = 1 \wedge \texttt{kz} = 0 \wedge \texttt{dx} = 0 \wedge (\texttt{dy} = 1 \vee \texttt{dy} = -1)) & \texttt{then } (\texttt{kx} = 1 \wedge \texttt{ky} = 0) \\ \texttt{elseif} & (\texttt{kdz} = 1 \wedge \texttt{kz} = 0 \wedge (\texttt{dx} = 1 \vee \texttt{dx} = -1) \wedge \texttt{dy} = 0) & \texttt{then } (\texttt{kx} = 0 \wedge \texttt{ky} = 1) \\ \texttt{else} & & (\texttt{kx} = 1 \wedge \texttt{ky} = 1) \end{cases}$$

where $\texttt{kx}$*,* $\texttt{ky}$*, and* $\texttt{kz}$ *are binary variables indicating whether the values of x, y, and z are needed.*

Proposition 4.12 describes how to model the XOR operation in guess-and-determine.

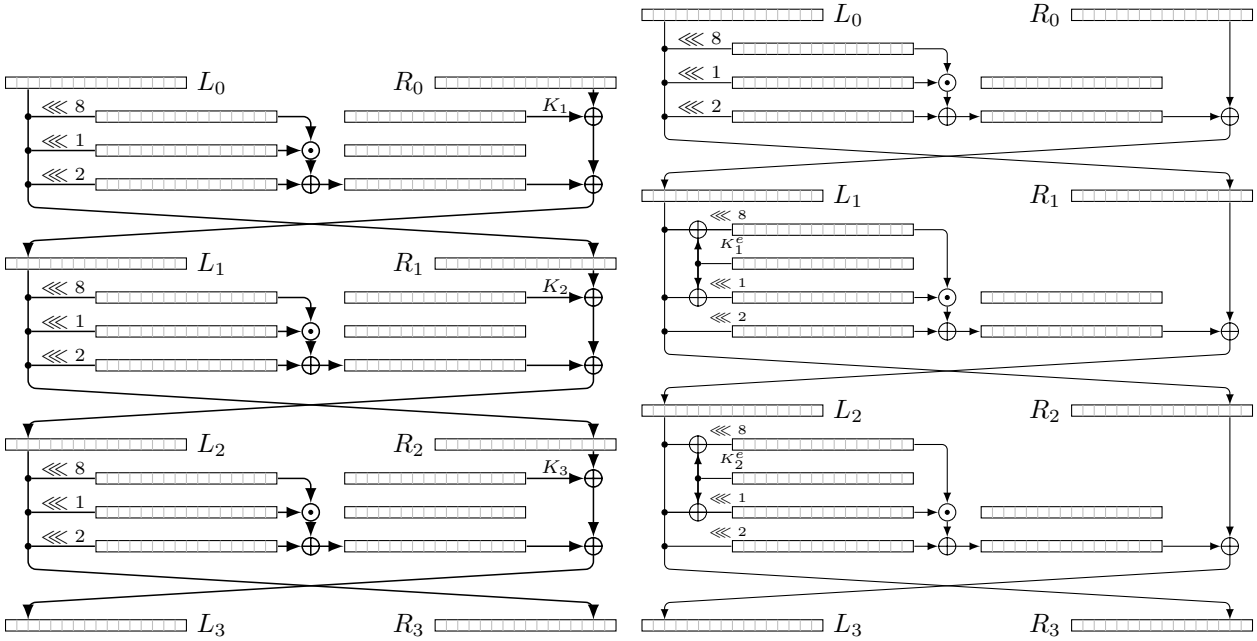
Proposition 4.12 ($\texttt{XOR}_2^{gd}$) *Suppose $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ is such that $z = f(x, y) = x \oplus y$. Let* $\texttt{kx}$*,* $\texttt{ky}$*, and* $\texttt{kz}$ *be binary variables indicating whether the values of x, y, and z are needed. Then, the valid propagations for detecting state cells whose values of the differences are needed through the* ***XOR*** *operation satisfy the following constraints:*

$$\texttt{XOR}_2^{gd}(\texttt{kz}, \texttt{kx}, \texttt{ky}, \texttt{dy}) := \begin{cases} \texttt{if} & (\texttt{kz} = 0) & \texttt{then } (\texttt{kx} = 0 \wedge \texttt{ky} = 0) \\ \texttt{else} & & (\texttt{kx} = 1 \wedge \texttt{ky} = 1) \end{cases}$$

Finally, we elucidate the method for identifying the key bits influencing the determination of data values. Consider $\texttt{ikb}_r[i] \in \{0, 1\}$, where $0 \leqslant i \leqslant (n-1)$ and $0 \leqslant r \leqslant (r_{\text{B}} - 1)$. This binary variable signifies whether the ith bit of the subkey in the rth round of E_{B} is involved. Let $\texttt{kxu}_r^0[i]$ and $\texttt{kxu}_r^1[i]$ be binary variables indicating whether the value of the i-th bit of the input of the r-th round function of E_{B} needs to be determined. Then, we can conclude that $\texttt{ikb}_r[i] = 1$ if and only if $\texttt{kxu}_{r+1}^0[i] = 1$. Otherwise $\texttt{ikb}_r[i] = 0$. Likewise, binary variables $\texttt{ikf}_r[i]$ are defined to represent the involved subkey bits in E_{F}. Now, Algorithm 5 outlines our CSP model for the guess-and-determine process using E_{B}. We denote $\texttt{CSP}_{\text{GD}}$ as our CSP models combining $\texttt{CSP}_{\text{B}}^{gd}$ and $\texttt{CSP}_{\text{F}}^{gd}$ for the guess-and-determine process across E_{B} and E_{F}.

Modeling Equivalent Subkey Technique.

The equivalent subkey technique has been widely used in various key-recovery attacks. This technique aims to reduce the number of guessed subkey bits by replacing the original subkeys with the equivalent subkeys. This technique was initially introduced by Isobe *et al.* [119] to investigate generic key recovery attacks on the Feistel scheme. Subsequently, it was adapted for zero-correlation attacks on SIMON [108], and for impossible differential and zero-correlation attacks on Simeck [97], [113].



(a) Original Subkey (b) Equivalent Subkey

Figure 4.5: Original Subkey vs Equivalent Subkey

To reduce the number of guessed subkey bits in the key recovery process, one can move the subkey K_i of the i-th round to the $(i+1)$-th round where $0 \leqslant i \leqslant (r_{\text{B}} - 1)$, to get the equivalent subkey K^e_{i+1} (please refer to Figure 4.7). Similarly, one can move the subkey K_i of the i-th round to the $(i-1)$-th round, for $(r_{\text{B}} + r_{\text{D}}) \leqslant i \leqslant (r_{\text{B}} + r_{\text{D}} + r_{\text{F}} - 1) = (r-1)$, to get the equivalent subkey K^e_{i-1} (please refer to Figure 4.7). This concept can be integrated into our key recovery model to minimize the number of implicated keys. For instance, the expressions of equivalent subkeys (Figure 4.5) can be formulated as follows:

$$\begin{cases} K^e_1 = K_0 \\ K^e_2 = (K^e_1 \lll 2) \oplus K_1 \\ K^e_3 = K^e_1 \oplus (K^e_2 \lll 2) \oplus K_2 \\ K^e_4 = K^e_2 \oplus (K^e_3 \lll 2) \oplus K_3 \\ \vdots \\ K^e_{r_{\text{B}}-1} = K^e_{r_{\text{B}}-3} \oplus (K^e_{r_{\text{B}}-2} \lll 2) \oplus K_{r_{\text{B}}-2} \\ K^e_{r_{\text{B}}} = K^e_{r_{\text{B}}-2} \oplus (K^e_{r_{\text{B}}-1} \lll 2) \oplus K_{r_{\text{B}}-1} \end{cases} \qquad \begin{cases} K^e_{r-2} = K_{r-1} \\ K^e_{r-3} = (K^e_{r-2} \lll 2) \oplus K_{r-2} \\ K^e_{r-4} = K^e_{r-2} \oplus (K^e_{r-3} \lll 2) \oplus K_{r-3} \\ K^e_{r-5} = K^e_{r-3} \oplus (K^e_{r-4} \lll 2) \oplus K_{r-4} \\ \vdots \\ K^e_{r_{\text{B}}+r_{\text{D}}} = K^e_{r_{\text{B}}+r_{\text{D}}+2} \oplus (K^e_{r_{\text{B}}+r_{\text{D}}+1} \lll 2) \oplus K_{r_{\text{B}}+r_{\text{D}}+1} \\ K^e_{r_{\text{B}}+r_{\text{D}}-1} = K^e_{r_{\text{B}}+r_{\text{D}}+1} \oplus (K^e_{r_{\text{B}}+r_{\text{D}}} \lll 2) \oplus K_{r_{\text{B}}+r_{\text{D}}} \end{cases}$$

For the r-th round of E_{B}, where $1 \leqslant r \leqslant r_{\text{B}} - 1$, let the two state variables where the equivalent subkey is xored be yu_r and zu_r. Furthermore, we define binary variables $\texttt{kyu}^0_r[i]$ and $\texttt{kzu}^0_r[i]$ to indicate whether the values of $yu_r[i]$ and $zu_r[i]$ are needed, and $\texttt{ikb}_r[i]$ to indicate whether the ith bit of the equivalent subkey in the r-th round of E_{B} is involved. Lastly, we apply the following constraints to implement the concept of the equivalent subkey

technique:

$$\texttt{CSP}^{ESK} := \begin{cases} \texttt{if } (\texttt{kyu1}_r^0[i] = 1 \vee \texttt{kzu1}_r^0[i] = 1) \texttt{ then} & (\texttt{ikb}_r[i] = 1) \text{ for } 0 \leqslant i \leqslant (n-1) \\ \texttt{else} & (\texttt{ikb}_r[i] = 0) \text{ for } 0 \leqslant i \leqslant (n-1) \end{cases}$$

where the variables $\texttt{kyu1}_r^0$ and $\texttt{kzu1}_r^0$ indicate the state after the α-bit right-rotation on $\texttt{kyu}^0[r]$ and after the β-bit rotation on $\texttt{kzu}^0[r]$, respectively (where $(\alpha, \beta) = (8, 1)$ for SIMON, and $(0, 5)$ for Simeck).

Important Observation. We can further reduce the number of involved subkey bits by considering a simple observation. Let $z = x \cdot y$, where x, y, z are binary variables. Also assume that $\texttt{kx}, \texttt{ky}, \texttt{kz}$ are binary variables to indicate whether the values of x, y, z are needed, and $\texttt{dx}, \texttt{dy}, \texttt{dz}$ are binary variables to indicate the truncated difference pattern in x, y, and z. Furthermore, assume x, y each involve some key bit information; for example, $x = x_1 \oplus k$ and $y = y_1 \oplus k'$, where k, k' are two key bits.

Consider the case $\texttt{dx} = \texttt{dy} = 0$ and $\texttt{kx} = \texttt{ky} = 1$. This implies we need to know the value of the state variables x, y, which normally means we need to guess two key bits k and k' in the basic model. However, this can be reduced to one bit of key information as follows:

- If we guess the key bit k such that $x = 0$, then $z = 0$, and we do not need to guess the value of k' as we don't need to know the value of y in this case.
- If we guess the key bit k such that $x = 1$, then $z = y$, and the key bit k' in y goes into z linearly. Then, it can usually be merged with the next key addition (to z) and does not need to be guessed separately at the moment.

This immediately implies that using the aforementioned idea, we can reduce the number of key bits involved. To optimize the complexity, we can choose whether guessing k or guessing k' is the better choice overall. We have included this technique in our automated model, which is easily integrated into the model with additional decision variables to optimize the choice of whether to guess k or k'.

Modeling the Complexity Formula

Given that the complexity formula of ID attacks includes some exponential terms and also the square root of some attack parameters, following the approach in [12], we reformulate them as follows to be able to incorporate them in our CP model: Suppose the number g represents the key bits retrieved during the guess-and-filter phase, with $P = 2^{-g}$. Assuming $P < \frac{1}{2}$, it follows that $1 < g \leqslant |k_{\text{B}} \cup k_{\text{F}}|$. Furthermore, under the approximation $(1 - 2^{(c_{\text{B}}+c_{\text{F}})})^N \approx e^{-N \cdot 2^{-(c_{\text{B}}+c_{\text{F}})}}$, we find $N = 2^{c_{\text{B}}+c_{\text{F}}+\log_2(g)-0.53}$. Also, define $\texttt{LG}(g) = \log_2(g) - 0.53$.

Thus, we can reframe the complexity analysis of the ID attack as follows:

$$
\begin{aligned}
T_0 &= \max\left\{\begin{array}{l}\min\limits_{\Delta\in\{\Delta_{\mathrm{B}},\Delta_{\mathrm{F}}\}}\{2^{\frac{c_{\mathrm{B}}+c_{\mathrm{F}}+n+1-|\Delta|+\mathtt{LG}(g)}{2}}\},\\ 2^{c_{\mathrm{B}}+c_{\mathrm{F}}+n+1-|\Delta_{\mathrm{B}}|-|\Delta_{\mathrm{F}}|+\mathtt{LG}(g)}\end{array}\right\}, \quad T_0 < 2^n \\
T_1 &= 2^{c_{\mathrm{B}}+c_{\mathrm{F}}+\mathtt{LG}(g)}, \qquad T_2 = 2^{|k_{\mathrm{B}}\cup k_{\mathrm{F}}|+\mathtt{LG}(g)}, \qquad T_3 = 2^{k-g} \\
T_{\mathrm{tot}} &= (T_0 + (T_1 + T_2)\, C_{E'} + T_3)\, C_E,\ T_{\mathrm{tot}} < 2^k \\
M_{\mathrm{tot}} &= \min\left\{2^{c_{\mathrm{B}}+c_{\mathrm{F}}+\mathtt{LG}(g)}, 2^{|k_{\mathrm{B}}\cup k_{\mathrm{F}}|}\right\},\ M_{\mathrm{tot}} < 2^k.
\end{aligned}
\tag{4.6}
$$

In this component, we discuss the aggregation of all CSP models and outline the approach to encode the complexity formula described in Equation 4.6. Hence, we express the complexity formulas from Equation 4.6 using the following constraints:

$$
\begin{cases}
\mathtt{d}_0 \coloneqq \mathtt{min}_{\Delta\in\{\Delta_{\mathrm{B}},\Delta_{\mathrm{F}}\}}\frac{1}{2}(\mathtt{c}_{\mathrm{B}} + \mathtt{c}_{\mathrm{F}} + \mathtt{block} - 1 - |\Delta| + \mathtt{LG}(g)),\\
\mathtt{d}_1 \coloneqq \mathtt{c}_{\mathrm{B}} + \mathtt{c}_{\mathrm{F}} + \mathtt{block} - 1 - |\Delta_{\mathrm{B}}| - |\Delta_{\mathrm{F}}| + \mathtt{LG}(g),\\
\mathtt{t}_0 \coloneqq \mathtt{max}\{\mathtt{d}_0, \mathtt{d}_1\},\ \mathtt{t}_0 < n\\
\mathtt{t}_1 \coloneqq \mathtt{c}_{\mathrm{B}} + \mathtt{c}_{\mathrm{F}} + \mathtt{LG}(g)\\
\mathtt{t}_2 \coloneqq |\mathtt{k}_{\mathrm{B}} \cup \mathtt{k}_{\mathrm{F}}| + \mathtt{LG}(g)\\
\mathtt{t}_3 \coloneqq \mathtt{k} - \mathtt{g},\\
\mathtt{T} \coloneqq \mathtt{max}\{\mathtt{t}_0, \mathtt{t}_1, \mathtt{t}_2, \mathtt{t}_3\}
\end{cases}
$$

Finally, our objective is to minimize $\mathtt{T}$. In our model, every variable is either binary or integer, with their domains restricted, except for $\mathtt{d}_0$, $\mathtt{d}_1$, and $\mathtt{t}_i$ for $i \in \{0, 1, 2, 3\}$, which are real numbers. Consequently, we have successfully modeled all pivotal parameters of the ID attack. Subsequently, we consolidate these CSP models into a cohesive framework and define an objective function aimed at minimizing the time complexity of the ID attack. We leverage the CP solver Or-Tools to optimize the discovery of ID attacks.

4.3.3 Application of Key Recovery Attack

We utilized our unified COP model to successfully identify a comprehensive ID attack across all variants of SIMON and Simeck. Table 4.2 summarizes the enhanced ID attacks achieved through our novel approach. As our analysis approach remains consistent across all the versions of SIMON and Simeck, we will detail an ID distinguisher and the corresponding key-recovery attack on SIMON-64-128 found using our model. The attack parameters for the other versions will be detailed in the following section.

13-round ID distinguisher of SIMON-64-128. Our unified CP model, as previously discussed, requires four integer parameters $(r_{\mathrm{B}}, r_{\mathrm{D}}, r_{\mathrm{M}}, r_{\mathrm{F}})$ which denote the length of the extended backward direction, the length of the distinguisher, the round of merging (in case of indirect contradiction), and the length of the extended forward direction. Using this framework, we discovered a 13-round ID distinguisher (based on direct contradiction) used to construct the optimized 23-round full ID attack on SIMON-64-128, as illustrated in Figure

4.6. In Figure 4.6, we notice that the truncated input difference Δ_{U} (illustrated in the upper triangle of $L_5||R_5$) where the upper triangle of the bits $L_5[0-4, 6-31]$ and all bits of R_5 have a constant difference (either 0 □ or 1 ◪), and the upper triangle of the bit $L_5[5]$ has an unknown difference (◪ we do not know whether there is a difference or not). Similarly, in the truncated output difference Δ_{L} (illustrated in the lower triangle of $L_{18}||R_{18}$), where the lower triangle of the bits $L_{18}[0-5, 6-31]$ and all the bits R_{18} have zero difference, and the lower triangle of the bit $L_{18}[6]$ has an unknown difference (◪). The 0-1 contradiction can be found in the bit $L_{10}[5]$ (equivalently $R_{11}[5]$).

It can be seen that this impossible differential is placed between rounds 5 and 18 and extended by $r_{\text{B}} = 5$ and $r_{\text{F}} = 5$ rounds in both directions. In this way, the first $23 = 5+13+5$ rounds of SIMON-64-128 were attacked.

23-round Complete ID Attack on SIMON-64-128. For the key recovery, the attack is illustrated in Figure 4.7. As discussed before, first we propagate the difference Δ_{U} (and Δ_{L}) through E_{B}^{-1} (and E_{F}) with probability one to obtain the truncated difference Δ_{B} (and Δ_{F}), where E_{B} and E_{F} represent the $r_{\text{B}} = 5$ rounds added before and $r_{\text{F}} = 5$ rounds after E_{D}, respectively. In this context, the bit difference one (active bit) and unknown bit difference are represented by ■ and ■. It can be seen that the truncated difference Δ_{B} is such that the bits $L_0[5, 12, 14-15, 21-22, 28, 30]$ and the bits $R_0[13, 20]$ have a constant difference (either □ 0 or 1 ■), which implies $|\Delta_{\text{B}}| = 54$ (the total number of non-fixed bit differences in Δ_{B}, illustrated by ■). Similarly, the truncated difference Δ_{F} is such that the bits $L_{23}[6-9, 13, 15]$ and the bits $R_{23}[5, 7-11, 14-17, 21, 23]$ have constant difference (either □ 0 or 1 ■). This means $|\Delta_{\text{F}}| = 46$. More precisely, by seeing Δ_{U} and Δ_{L}, it can be asserted that there are two output patterns that produce the longest impossible differential for given input patterns, and as $|\Delta_{\text{U}}| = 2$, then there are two possible input patterns. Using the idea of multiple differentials discussed in [44], we can update $|\Delta_{\text{B}}| = 55$ and $|\Delta_{\text{F}}| = 47$. In the context, c_{B} and c_{F} are typically referred to as the number of bit filters (bit conditions, or we can say probabilistic transitions) that should be satisfied for differential propagations $\Delta_{\text{B}} \rightarrow \Delta_{\text{U}}$ and $\Delta_{\text{L}} \leftarrow \Delta_{\text{F}}$, respectively. In Figure 4.7, the position of bit conditions is represented by □ (which means in our model, the values of the variable `cb` corresponding to those positions are 1). Thus, the differential transition $\Delta_{\text{B}} \rightarrow \Delta_{\text{U}}$ requires satisfaction of $c_{\text{B}} = 53$ filters, while $\Delta_{\text{L}} \leftarrow \Delta_{\text{F}}$ necessitates satisfaction of $c_{\text{F}} = 45$ filters.

As previously mentioned, it is necessary to identify the state bits whose differences, values, or both are required to verify bit conditions in $\Delta_{\text{B}} \xrightarrow{E_{\text{B}}} \Delta_{\text{U}}$ and $\Delta_{\text{L}} \xleftarrow{E_{\text{F}}^{-1}} \Delta_{\text{L}}$. In Figure 4.7, the bit position where the value is needed and the difference is required for checking bit conditions are represented by ◹ (which means in our model, the value of the variable `kx` corresponding to that bit position is 1) and ▨ (which means in our model, the value of the variable `kdx` corresponding to that bit position is 1), respectively. Finally, we know that the key bits $k_{\text{B}} \cup k_{\text{F}}$ are involved in deriving the difference Δ_{U} and Δ_{L} from Δ_{B} and Δ_{F}, respectively. In Figure 4.7, it can be seen that $k_{\text{B}} = 65$ and $k_{\text{F}} = 57$, which means the total number of subkey bits involved is 122.

Finally, we can have the complexities of this attack by using the complexity formula discussed in Equation 4.6, which can be modeled in our framework by the constraints discussed

in Subsection 4.3.2. Therefore, the complexities of our attack are:

$$\begin{cases} \texttt{Time Complexity} : 2^{124} \\ \texttt{Data Complexity} : 2^{62.47} \\ \texttt{Memory Complexity} : 2^{99.5} \end{cases}$$

The previous best ID attack is found on 22-round SIMON-64-128 [44], [93].

Discussion. The gap between the actual time complexity and the output of Boura *et al.* 's formula [44] is well known. This formula provides a lower bound for the complexity of the guess-and-filter step, which is typically close to the actual value. The reason that [12], [13] and we are using this formula to build a unified CP model has been discussed in [13] (see the last paragraph of Section 2.1 of [13]). Briefly, there are two approaches: using this formula to estimate the complexity (as done in most of the previous works) and keeping the model very easy to solve, or incorporating all the details of the step-by-step early-abort technique into the CP model that is more accurate but makes the model very hard to solve. The first approach is relatively fast and mostly returns an optimum attack quickly. The second one can theoretically find the exact time complexity, but it is tough to solve because the early-abort technique is a multistep guess-and-determine procedure; implementing it into the CP model makes it very hard to solve. So, we chose the first one to find a nearly optimum attack and then check whether the actual time complexity matches the formula's output discussed in [44]. In this regard, we found that sometimes the complexities deduced from Boura *et al.* 's formula [44] are lower (but relatively close) than those inferred from the step-by-step early abort technique. For example, in Simon-64-128, we performed the step-by-step early abort technique and got the actual complexities as follows:

$$\begin{cases} \texttt{Time Complexity} : 2^{126.94} \\ \texttt{Data Complexity} : 2^{62.47} \\ \texttt{Memory Complexity} : 2^{98.47} \end{cases}$$

4.4 Applications

This section discusses the application of our method to both ARX and AndRX ciphers. ARX ciphers are based on three primary operations: modular addition ($x \boxplus y$), bitwise rotation ($x \lll n$), and XOR ($x \oplus y$). We analyze different ARX constructions used in block ciphers (*e.g.*, LEA and SPECK), stream ciphers (*e.g.*, ChaCha [15]) and MAC algorithms (*e.g.*, SipHash and Chaskey). Furthermore, we expand our analysis to AndRX ciphers, which replace modular addition with bitwise AND ($x \odot y$) operations, as utilized in ciphers such as SIMON and Simeck. Refer to Section 2.4 for brief specifications of these ciphers.

4.4.1 Application to ARX Ciphers

We have successfully applied our bit-wise CP-based model to find ID distinguishers based on satisfiability for several ARX ciphers: all versions of SPECK, LEA, ChaCha, Chaskey, and

SipHash. Table 4.1 presents the ID distinguishers identified using our tool and compares them with previous ID distinguishers. The subsequent sections provide detailed explanations of the attacks on each ARX cipher.

Application To SPECK

SPECK is a family of ARX ciphers with a generalized Feistel structure, known for its excellent performance in both hardware and software implementations. For more details on the specification, please refer to Section 2.4.3. Since its publication, numerous cryptanalysi s have been conducted [120]–[122]. In this context, we discuss the impact of impossible differential cryptanalysis on SPECK.

We applied our method to search for ID distinguishers for all versions of SPECK. Our findings include 6-round ID distinguishers for SPECK-32, SPECK-48, and SPECK-64, matching the previous best-known ID distinguishers. Specifically, the authors of [98] analyzed the differential properties of SPECK's round function and transformed these properties into Boolean expressions to construct a SAT model for searching impossible differentials. They identified 3, 20, and 157 6-round IDs for SPECK-32, SPECK-48, and SPECK-64, respectively, with Hamming weight one in both input and output differences. In contrast, our tool found clusters of 2^4, 2^{17}, and 2^{33} ID distinguishers for these versions with Hamming weights of at least one in both input and output differences. These clusters were discovered in a single run of our tool, demonstrating the advantage of our framework over previous methods, where finding such large clusters would be difficult or infeasible. Additionally, we report 6-round ID distinguishers for SPECK-96 and SPECK-128 for the first time, discovering clusters of 2^{65} and 2^{97} IDs, respectively (for example, Figure 4.8).

Application to ChaCha

We applied our method to search for an ID distinguisher for ChaCha256. For a brief specification of ChaCha, please refer to Section 2.4.5. The most popular cryptanalysis of ChaCha relies on differential cryptanalysis using probabilistic neutral bits (PNBs). Since 2008 [123], there have been several advancements in PNB-based attacks on ChaCha, resulting in improved attacks. Researchers have traditionally focused on incorporating single-bit differences at the beginning of differential-linear distinguishers for key-recovery attacks on ChaCha. Recently, [124] introduced an innovative approach with a 5-round differential-linear distinguisher considering 2-bit differences at the beginning. This 5-round distinguisher, integrated with the PNB framework, led to an enhanced key-recovery attack specifically tailored for a 7-round ChaCha cipher.

To the best of our knowledge, no cryptanalytic results have been reported regarding ID distinguishers for ChaCha. Using our model, we have detected a cluster of 2^{80} ID distinguishers for 5-round ChaCha for the first time. For detailed results, please refer to Table 4.5.

Application to Chaskey

Chaskey is a permutation-based MAC algorithm presented by Mouha *et al.* in 2014 [16]. Section 2.4.7 briefly describes the specification of Chaskey. Several attacks have been per-

formed on Chaskey such as rotational cryptanalysis [125], differential-linear cryptanalysis on 7-round Chaskey [126], and impossible differential cryptanalysis [100]. More precisely, the authors of [100] used the MILP model proposed by [95] to search for impossible differentials by fixing the input/output differential. This approach led to the discovery of 15 4-round ID distinguishers for Chaskey.

We applied our advanced bit-wise CP model to find ID distinguishers of Chaskey. Our model detected a cluster of 2^7 ID distinguishers for the 4-round Chaskey in a single run, aligning with the best previous results in terms of the number of rounds. For detailed results, please refer to Table 4.6.

Application to LEA

LEA (Lightweight Encryption Algorithm) [18] is a block cipher developed by the Korea Internet & Security Agency (KISA) to provide lightweight encryption in resource-constrained environments. Please refer to Section 2.4.4 for a concise specification of LEA. The designers of LEA [18], conducted differential cryptanalysis on the LEA cipher and identified a differential path extending up to 11 rounds with a probability of 2^{-98}. Additionally, they performed differential-linear, impossible differential, and linear cryptanalysis, presenting paths for 14, 10, and 11 rounds, respectively. In [127], the authors proposed 9-round ZC distinguishers for LEA. Additionally, in [95], the authors applied their MILP model to search for ID distinguishers of LEA and reported a 10-round ID distinguisher.

We employed our advanced model to search for an ID distinguisher for LEA and identified a cluster of 2^2 ID distinguishers for the 10-round LEA in a single run, matching the best previous results in terms of the number of rounds. For detailed results, please see Table 4.7.

Applications to SipHash

SipHash is a family of pseudorandom functions introduced by Aumasson and Bernstein at Indocrypt 2012 [17], designed for short message inputs. Please refer to Section 2.4.6 for a brief specification of SipHash. In [128], the authors extended their methods to compute the probability of ARX functions, resulting in a characteristic for SipHash-2-4 with a probability of $2^{-236.3}$ and a distinguisher for the finalization of SipHash-2-4 with practical complexity. Other significant advancements in cryptanalysis of SipHash are documented in [129], [130]. Using our advanced model, we conducted a search for an ID distinguisher for SipHash. To our knowledge, we are the first to identify a group of 2^{14} ID distinguishers for the 4-round version of SipHash using our approach. Please refer to Table 4.8 for the detailed result.

In summary, within a few minutes, our bit-wise CP model can produce the best-known ID distinguishers for the targeted ARX ciphers with large block sizes.

4.4.2 Application to AndRX Ciphers

This chapter begins by introducing advanced bit-wise CP models designed to detect ID and ZC distinguishers in AndRX ciphers such as SIMON and Simeck. Additionally, we expand the ID distinguisher model to create a unified COP model for generating full ID attacks on AndRX ciphers. Please refer to Section 2.4.1, and 2.4.2 for a brief overview

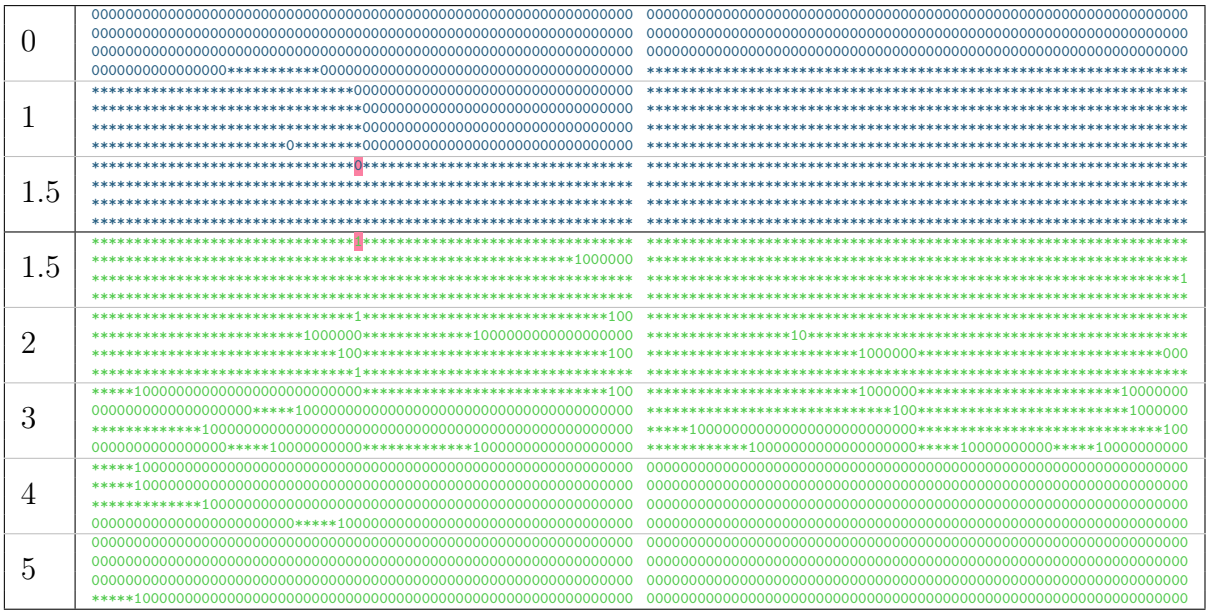


Table 4.5: Cluster of 2^{80} impossible-differential distinguishers for 5-round ChaCha.

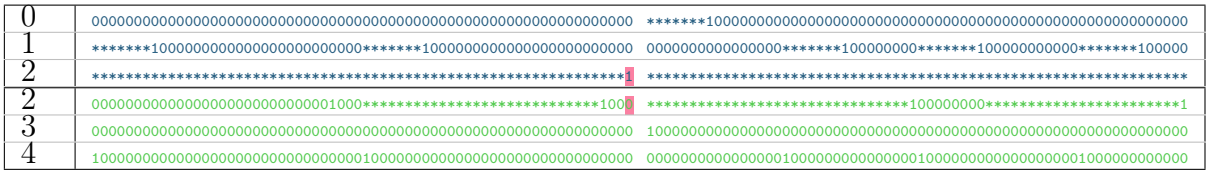


Table 4.6: Cluster of 2^{7} impossible-differential distinguishers for 4-round Chaskey.

Table 4.7: Cluster of 2^{2} impossible-differential distinguishers for 10-round LEA.

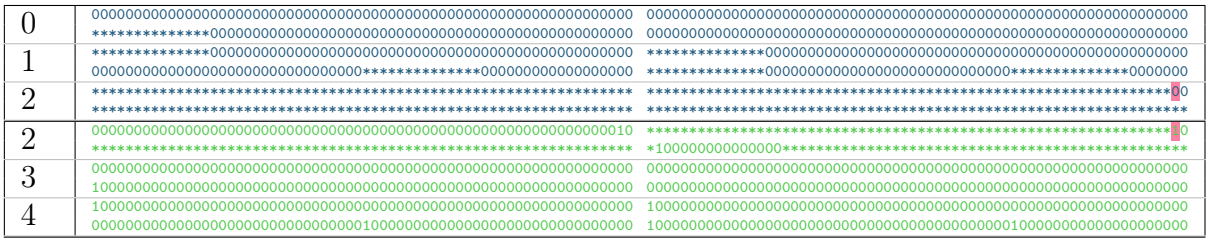


Table 4.8: Cluster of 2^{14} impossible-differential distinguishers for 4-round SipHash.

of the specifications of SIMON and Simeck, respectively. Moreover, we demonstrated most of the existing cryptanalytic attacks on SIMON and Simeck in Table 4.3 and Table 4.4, respectively. It is important to note that the best existing cryptanalysis of both ciphers is linear cryptanalysis. Thus, we do not claim that ID attacks are the best cryptanalytic results on SIMON and Simeck. Instead, our contribution lies in developing an efficient tool to find full ID attacks on these ciphers. Our tool provides a novel solution that eliminates the need for manual contradiction discovery or input-output fixing, as required by previous automated methods. The details of our ID and ZC attacks on SIMON, and Simeck are as follows.

Table 4.9: Summary of our ZC distinguishers for AndRX ciphers

Cipher	(r_D, r_M)	Contradiction (round/type)	Ref.
SIMON-32	$(11, -)$	6/Direct	[108]
	$(11, 6)$	6/Direct	This chapter
SIMON-48	$(12, -)$	7/Direct	[108]
	$(12, 7)$	7/Direct	This chapter
SIMON-64	$(13, -)$	8/Direct	[108]
	$(13, 8)$	8/Direct	This chapter
SIMON-96	$(16, -)$	10/Direct	[108]
	$(16, 10)$	10/Direct	This chapter
SIMON-128	$(19, -)$	12/Direct	[108]
	$(19, 12)$	12/Direct	This chapter
Simeck-32	$(11, -)$	−/Direct	[131]
	$(11, 5)$	5/Direct	This chapter
Simeck-48	$(15, -)$	2/Indirect	[97]
	$(15, 8)$	2/Indirect	This chapter
Simeck-64	$(17, -)$	2/Indirect	[97]
	$(17, 8)$	2/Indirect	This chapter

Applications to SIMON

First, we applied our improved bit-wise CP models for ID distinguishers on all variants of SIMON. The results include clusters of 2^4, 2^7, 2^{20}, 2^{33} and 2^{46} ID distinguishers for 11-round SIMON-32, 13-round SIMON-64, 16-round SIMON-96 and 19-round SIMON-128, respectively. These results match the previously best-known ID distinguishers [44] in terms of number of rounds. In [44], the authors used a miss-in-the-middle approach to find ID distinguishers, relying on manual methods to identify contradictions and fix the input/output differences. In contrast, our tool does not fix input/output differences, allowing us to discover a large cluster of IDs in a single run. Moreover, an important distinction is that most of the ID distinguishers previously reported in [44] have a Hamming weight of one in both input and output differences. Our tool, however, identifies ID distinguishers with Hamming weights

greater than one in both input and output differences. This capability underscores the significant contribution of our approach, offering more comprehensive and efficient discovery of ID distinguishers.

Furthermore, we applied our bit-wise model for ZC distinguishers to all versions of SIMON and produced ZC distinguishers of 11-round, 12-round, 13-round, 16-round, and 19-round SIMON-32, SIMON-48, SIMON-64, SIMON-96, and SIMON-128, respectively. Although no new results were found for SIMON regarding ID and ZC distinguishers, our automated tool can identify several ID and ZC distinguishers in just one run within a few minutes. Please refer to Table 4.9 for more details of the attack parameters for ZC distinguishers of SIMON, and Table 4.10 for ID distinguishers of SIMON.

Table 4.10: Summary of our ID attack parameters for AndRX ciphers. r_{D}: The length of the distinguisher. r_{B}: The length of the extended backward direction. r_{F}: The length of the extended forward direction. r_{M}: The position of merging

Cipher	$(r_{\mathrm{B}}, r_{\mathrm{D}}, r_{\mathrm{F}}, r_{\mathrm{M}})$	Contradiction (round/type)	$(c_{\mathrm{B}}, c_{\mathrm{F}})$	$(\lvert\Delta_{\mathrm{B}}\rvert, \lvert\Delta_{\mathrm{F}}\rvert)$	$\lvert k_{\mathrm{B}} \cup k_{\mathrm{F}}\rvert$	g
SIMON-32-64	$(3, 11, 6, 5)$	5/ Direct	$(14, 30)$	$(14, 32)$	61	2
SIMON-48-72	$(4, 12, 4, 6)$	6/ Direct	$(34, 30)$	$(38, 32)$	64	5
SIMON-48-96	$(4, 12, 5, 6)$	6/ Direct	$(34, 40)$	$(38, 42)$	86	7
SIMON-64-96	$(4, 13, 4, 6)$	6/ Direct	$(35, 31)$	$(39, 33)$	64	10
SIMON-64-96	$(4, 13, 5, 6)$	6/ Direct	$(37, 44)$	$(41, 46)$	90	4
SIMON-64-128	$(4, 13, 5, 6)$	6/ Direct	$(38, 44)$	$(44, 46)$	93	25
SIMON-64-128	$(5, 13, 5, 6)$	6/ Direct	$(53, 45)$	$(55, 47)$	122	4
SIMON-96-96	$(4, 16, 4, 9)$	9/ Direct	$(32, 36)$	$(34, 40)$	90	4
SIMON-96-144	$(5, 16, 4, 9)$	9/ Direct	$(48, 36)$	$(50, 40)$	121	19
SIMON-128-128	$(4, 19, 5, 8)$	8/ Direct	$(34, 48)$	$(54, 50)$	111	21
SIMON-128-192	$(5, 19, 5, 8)$	8/ Direct	$(45, 48)$	$(49, 50)$	140	25
SIMON-128-192	$(5, 19, 6, 8)$	8/ Direct	$(45, 64)$	$(47, 66)$	174	11
SIMON-128-256	$(5, 19, 6, 8)$	8/ Direct	$(56, 64)$	$(62, 66)$	194	24
SIMON-128-256	$(6, 19, 6, 8)$	8/ Direct	$(61, 64)$	$(65, 66)$	243	8
Simeck-32-64	$(5, 11, 4, 5)$	5/ Direct	$(27, 21)$	$(31, 23)$	53	8
Simeck-48-96	$(5, 15, 5, 7)$	13/ indirect	$(33, 33)$	$(35, 35)$	91	3
Simeck-64-128	$(5, 17, 5, 9)$	1/ indirect	$(34, 34)$	$(35, 35)$	101	2

Key-recovery We applied our unified COP model to develop a full ID attack on all variants of SIMON in single key settings. The parameters of our ID attack are detailed in Table 4.10, including input parameters $(r_{\mathrm{B}}, r_{\mathrm{D}}, r_{\mathrm{F}}, r_{\mathrm{M}})$ and output parameters $(c_{\mathrm{B}}, c_{\mathrm{F}}, |\Delta_{\mathrm{B}}|, |\Delta_{\mathrm{F}}|$, the total involved keys as $|k_{\mathrm{B}} \cup k_{\mathrm{F}}|$, $g)$, crucial for generating the complexities of the ID attacks, as shown in Equation 4.6. As detailed in Table 4.2, our ID attack for SIMON demonstrated significant improvements:

- A 22-round complete ID attack on SIMON-64-96
- A 23-round complete ID attack on SIMON-64-128

- A 28-round complete ID attack on SIMON-128-128
- A 31-round complete ID attack on SIMON-128-256

Each of these attacks extends the number of rounds by one compared to the previous best ID attacks on the respective versions. Furthermore, we achieved a 30-round full ID attack on SIMON-128-192 (Figure 4.10 in Section 4.6), which surpasses the previous best ID attacks by two rounds. For a detailed comparison of our attacks with the previous best attacks, please refer to Table 4.2. While we did not improve the number of rounds for other versions of SIMON, we significantly improved the time complexity of the attacks. Moreover, it is noteworthy that while most of the previous attacks on SIMON were full codebook attacks, our COP model enabled us to devise ID attacks that are not full codebook attacks for most SIMON variants. This is a notable achievement of our COP model. Most previous full ID attacks on SIMON relied on manual approaches. In contrast, [93] introduced an automated tool to find ID key recovery attacks on SIMON, though their tool required several days to produce results for larger versions. In comparison, our unified COP model offers a more efficient and streamlined approach. A detailed comparison of our method and the previous approaches is presented in the subsequent section, highlighting the advancements and efficiency of our contributions.

Applications to Simeck

Initially, we employed our bit-wise CP models for ID and ZC distinguishers, across all variants of Simeck. As a result of this, we construct 11-, 15- (indirect), and 17-round (indirect) ID and ZC distinguishers of Simeck-32, Simeck-48, Simeck-64, respectively. All the distinguishers are in accordance with the best previous ID and ZC distinguishers of Simeck.

Key Recovery. We utilized our integrated COP model to execute a comprehensive ID attack across all variants of Simeck in single-key configurations. Please refer to Table 4.10 for details of our ID attack parameters on all versions of Simeck. The authors of [103] proposed 20-, 25-, and 27-round ID attacks on Simeck-32, Simeck-48, and Simeck-64 with time complexities of $2^{61.11}$, $2^{94.23}$, and $2^{126.56}$, respectively (for Simeck-32, please refer to Figure 4.11, and for Simeck-64, please refer to Figure 4.12). The authors in [103] presented their attack, but did not explain anything regarding their attack algorithm. However, we found 20-, 25-, and 27-round ID attacks for the corresponding variants of Simeck with time complexities of $2^{55.79}$, $2^{93.05}$, and 2^{126}, respectively. Interestingly, we discovered a 20-round ID attack on Simeck-32 with data complexity $2^{29.79}$, while the previous best attack on Simeck-32 was a full codebook attack. For a more detailed comparison of our ID attacks with previous works, please refer to Table 4.2.

Discussion

Table 4.3 and Table 4.4 provide an overview of the most popular existing cryptanalytic results on SIMON and Simeck. Table 4.2 summarizes all ID attacks constructed using our tool and compares them with the existing ID attacks on SIMON and Simeck. In [44], the authors followed a manual approach to find ID attacks on all versions of SIMON, whereas in [93],

the authors introduced an automatic tool to find ID attacks on various ciphers, including SIMON. We strongly believe that the performance of our tool significantly surpasses the tool presented in [93]. While the tool in [93] requires several days to generate attacks on variants of SIMON with the largest block size, our tool achieves enhanced results within hours, even when executed on a regular laptop. Additionally, the authors of [93] did not provide detailed attack descriptions for larger variants of SIMON. Our tool offers several other advantages, such as its applicability for zero-correlation attacks and its ability to leverage any state-of-the-art CP/SMT/MILP solvers, unlike the tool in [93]. Another key advantage of our tool lies in its foundation on a unified optimization problem that considers all attack parameters, in contrast to the approach in [93], which involves enumerating numerous ID distinguishers within a limited search space and then selecting the optimal one. This fundamental difference contributes significantly to the speed of our tool, allowing us to readily apply it to large variants of SIMON and Simeck. It is important to recognize the challenges encountered when constructing a unified CP model to identify a complete ID attack on ARX ciphers. The primary difficulty arises in modeling the guess-and-determine process in the outer sections, particularly when dealing with state bits whose differences, values, or both are essential for verifying bit conditions through modular additions. Consequently, we have decided to leave this aspect for future exploration.

4.5 Conclusion

This chapter enhanced the CP model for discovering complete ID attacks in [12], [13] from three significant perspectives. Firstly, we expanded the bit-wise CP model of [13] to encompass ARX and AndRX designs. Additionally, we demonstrated how to extend this model to detect ZC distinguishers. Moreover, we introduced a novel CP model for finding ID distinguishers based on direct and indirect contradictions. This model is generic and not limited to ARX and AndRX designs. Subsequently, we extended the CP model for key recovery of ID attacks in [12] to encompass bit-oriented designs, particularly AndRX designs. Finally, we integrated our new CP model for detecting ID distinguishers with the CP model for key recovery, proposing a unified CP model for identifying complete ID attacks.

4.6 Related Figures

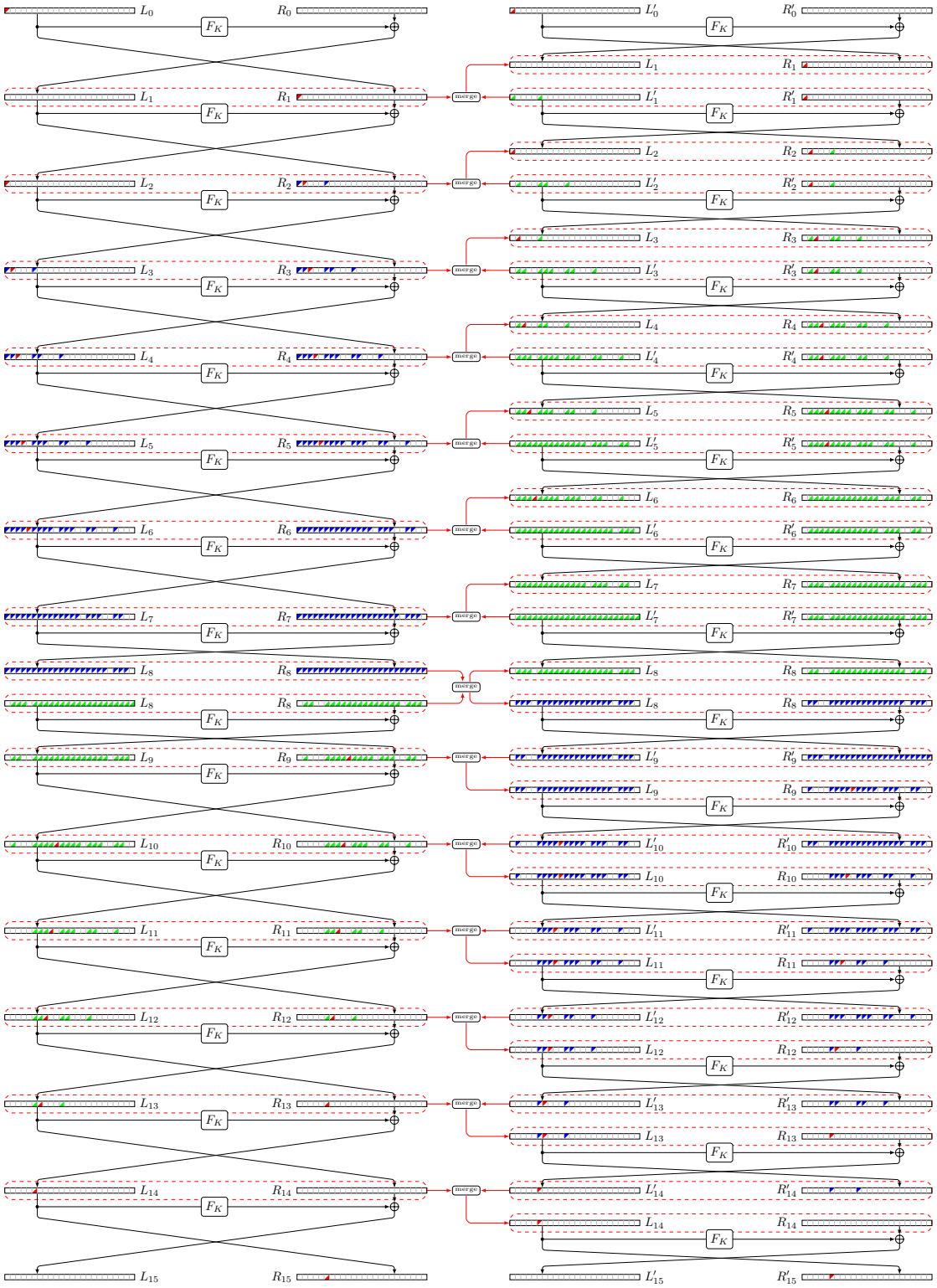
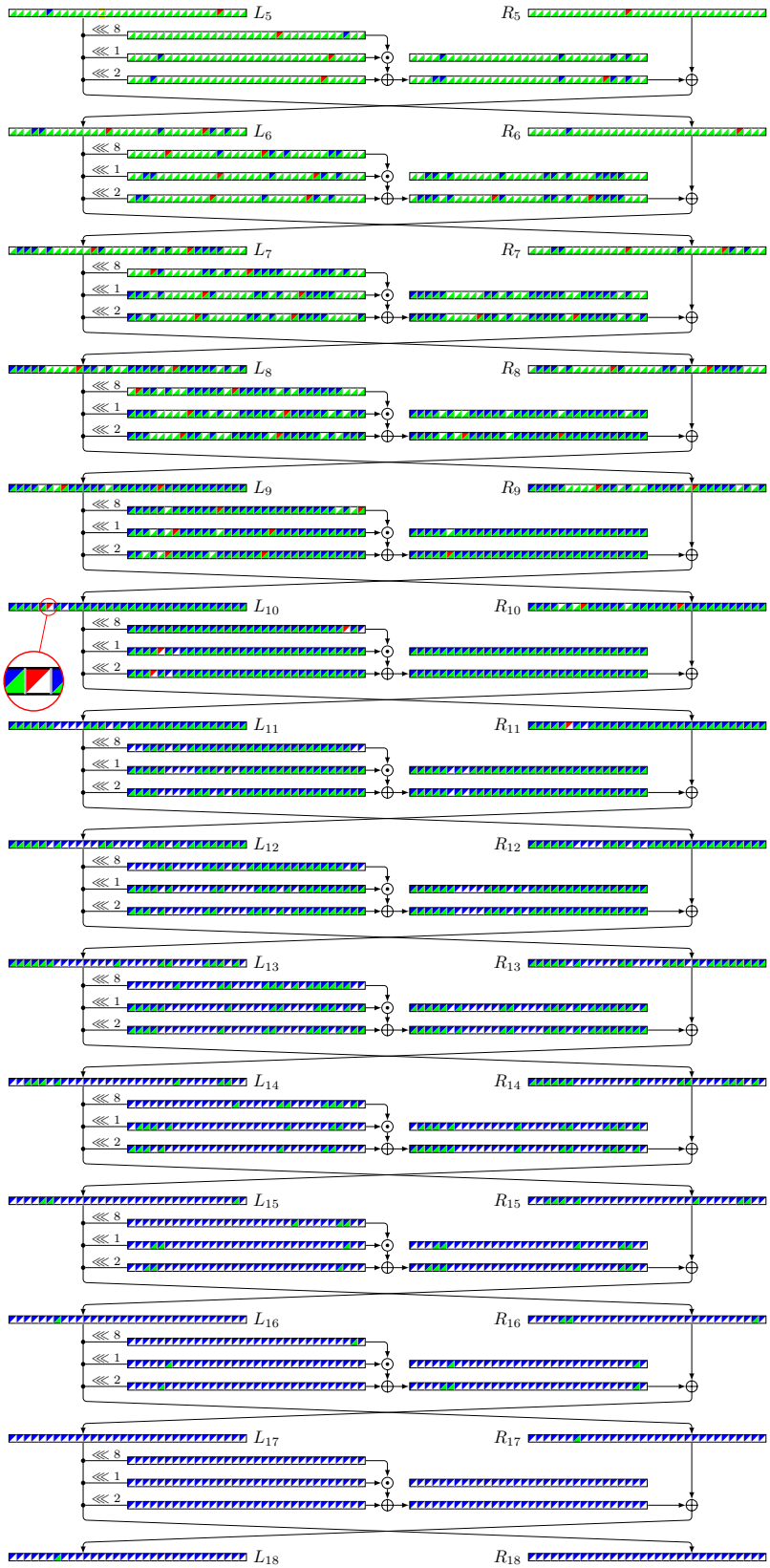
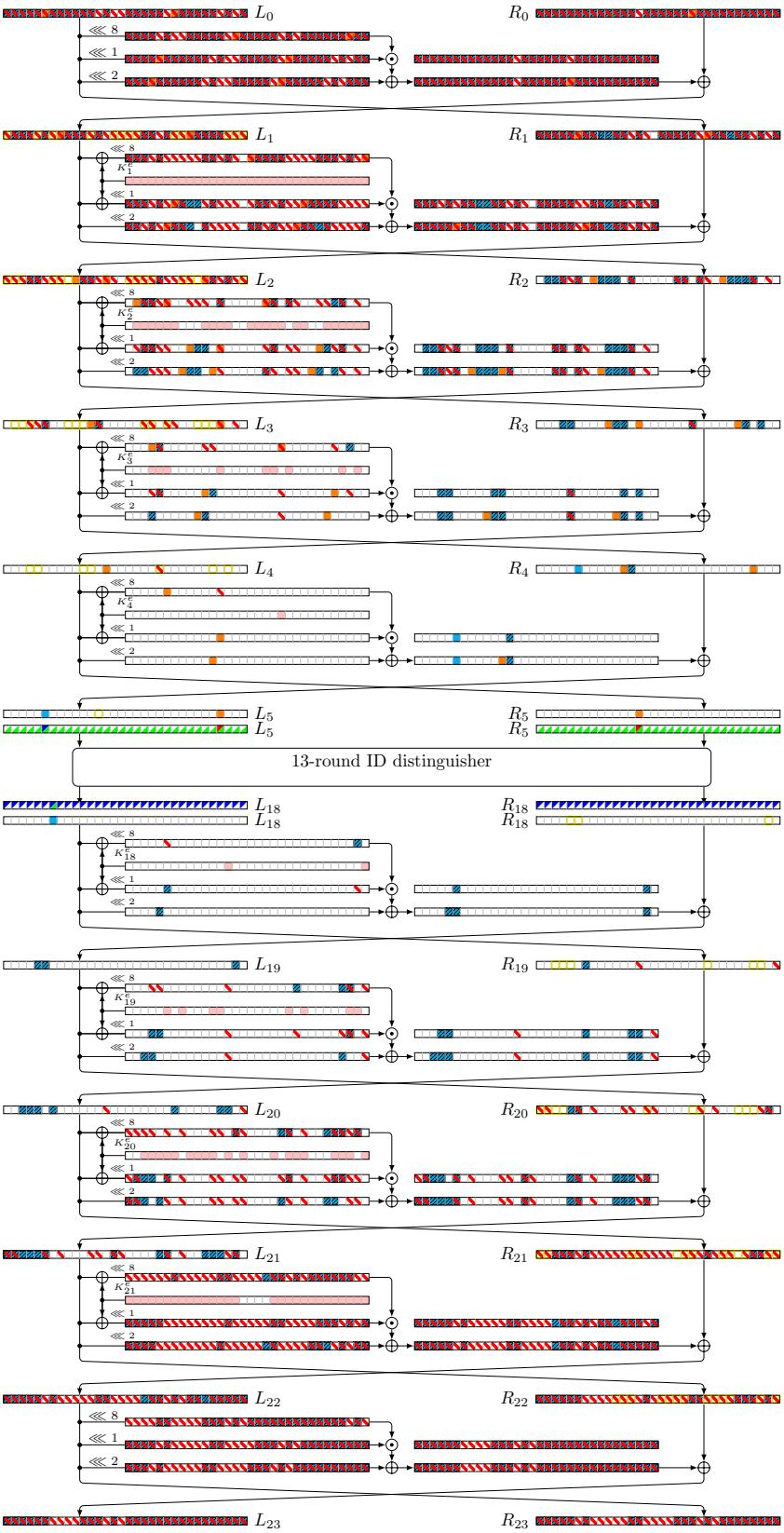


Figure 4.4: 15-round (indirect) ZC distinguisher for Simeck48. In this case, the bit difference in the upper triangle of $L_2[0]$ (in the left-hand column) is 1, whereas the bit difference in the lower triangle of $L'_2[0]$ is 0. This leads to a contradiction occurring in the second round.



bit difference (linear mask) 1 forward
bit difference (linear mask) 1 backward
unknown difference (linear mask) forward
unknown difference (linear mask) backward

Figure 4.6: 13-round ID distinguisher for attack on 23-round SIMON64-128.



1 any difference is needed value is needed involved in the key recovery filter

Figure 4.7: Key recovery of the attack on 23-round SIMON64-128.

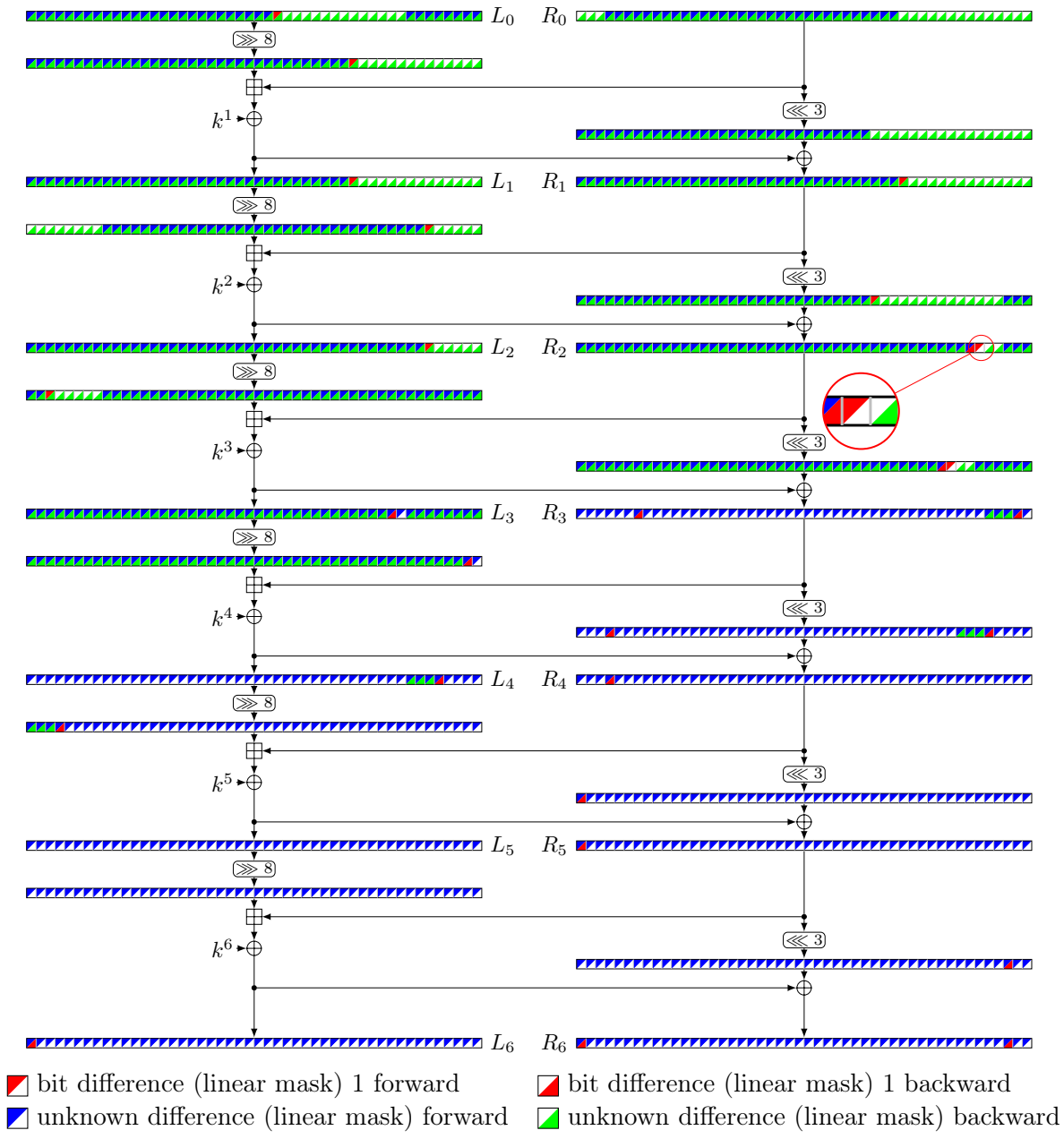


Figure 4.8: Cluster of 2^{65} ID distinguishers for 6-round SPECK-96.

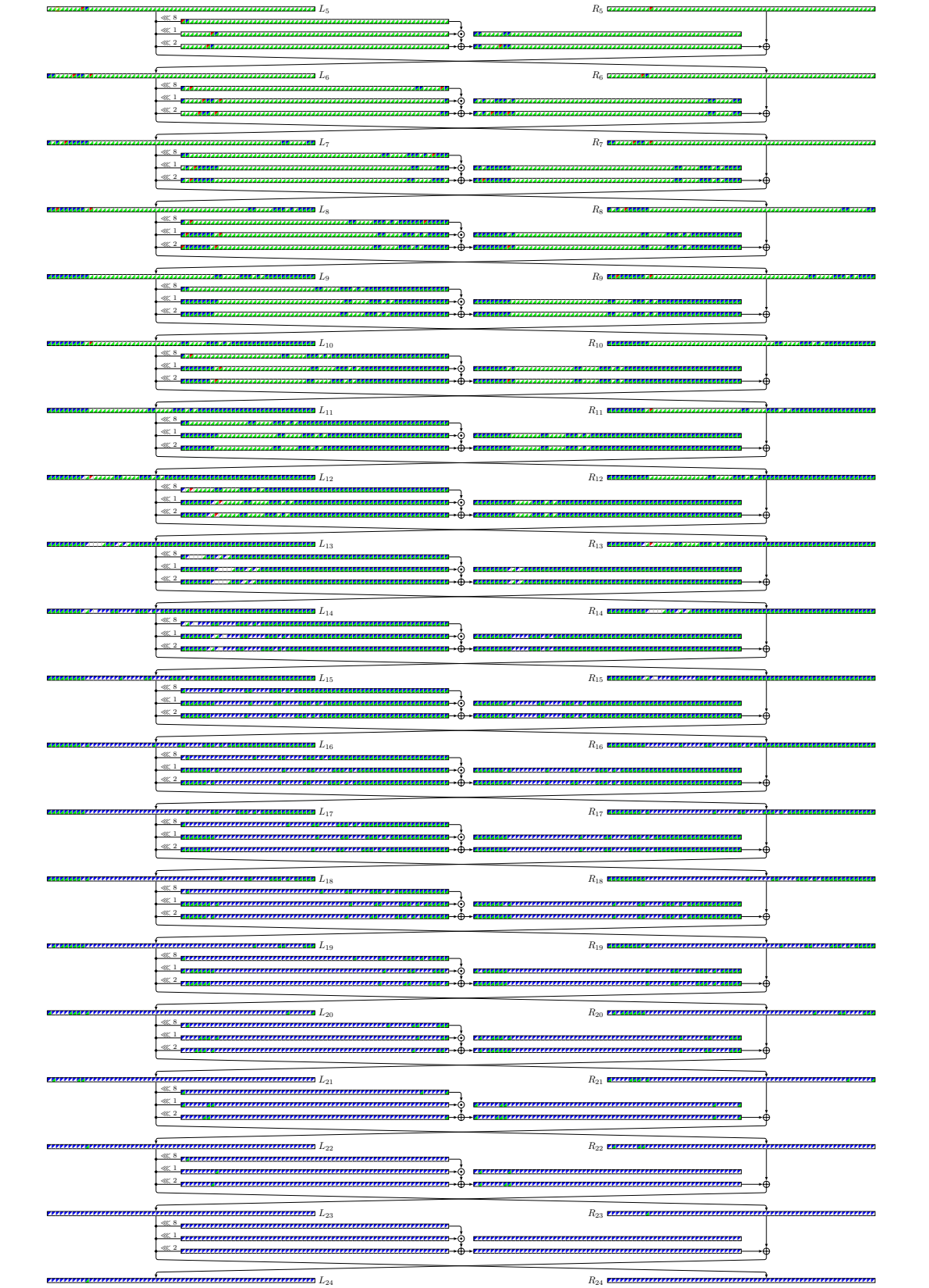


Figure 4.9: 19-round ID distinguisher for attack on 30-round SIMON128-192.

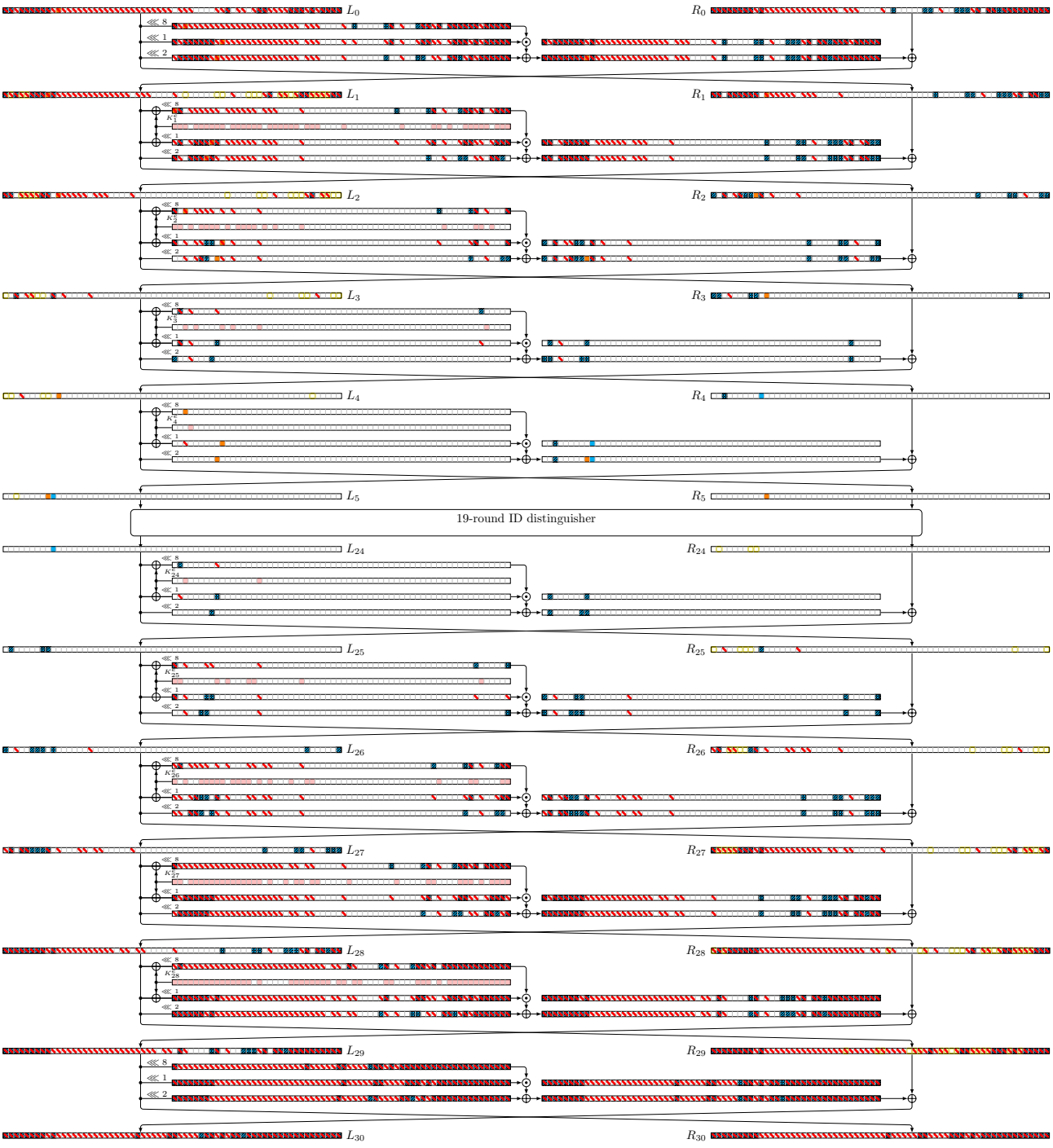


Figure 4.10: Key recovery attack on 30-round SIMON128-192.

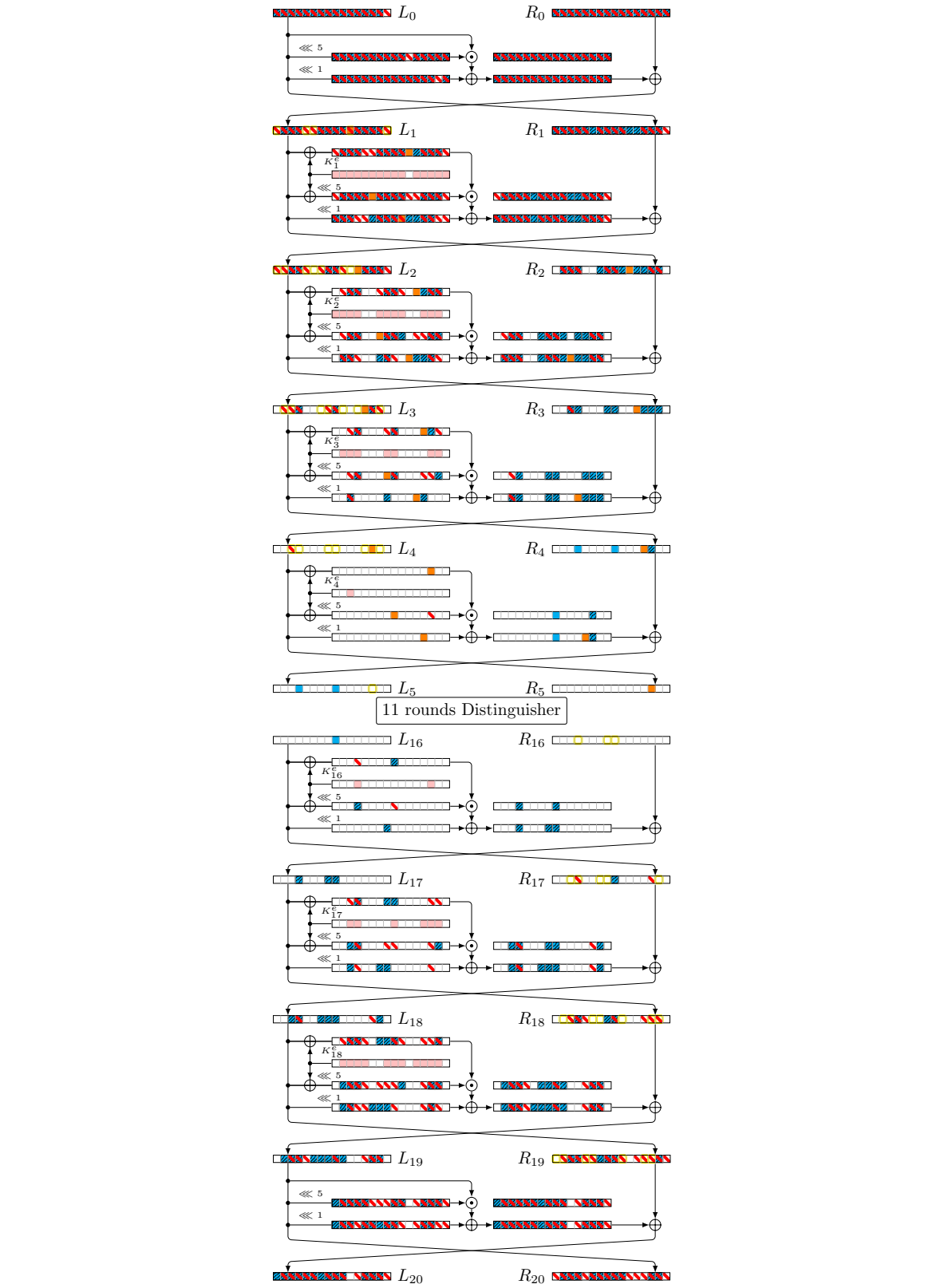


Figure 4.11: Key recovery attack on 20-round Simeck32-64.

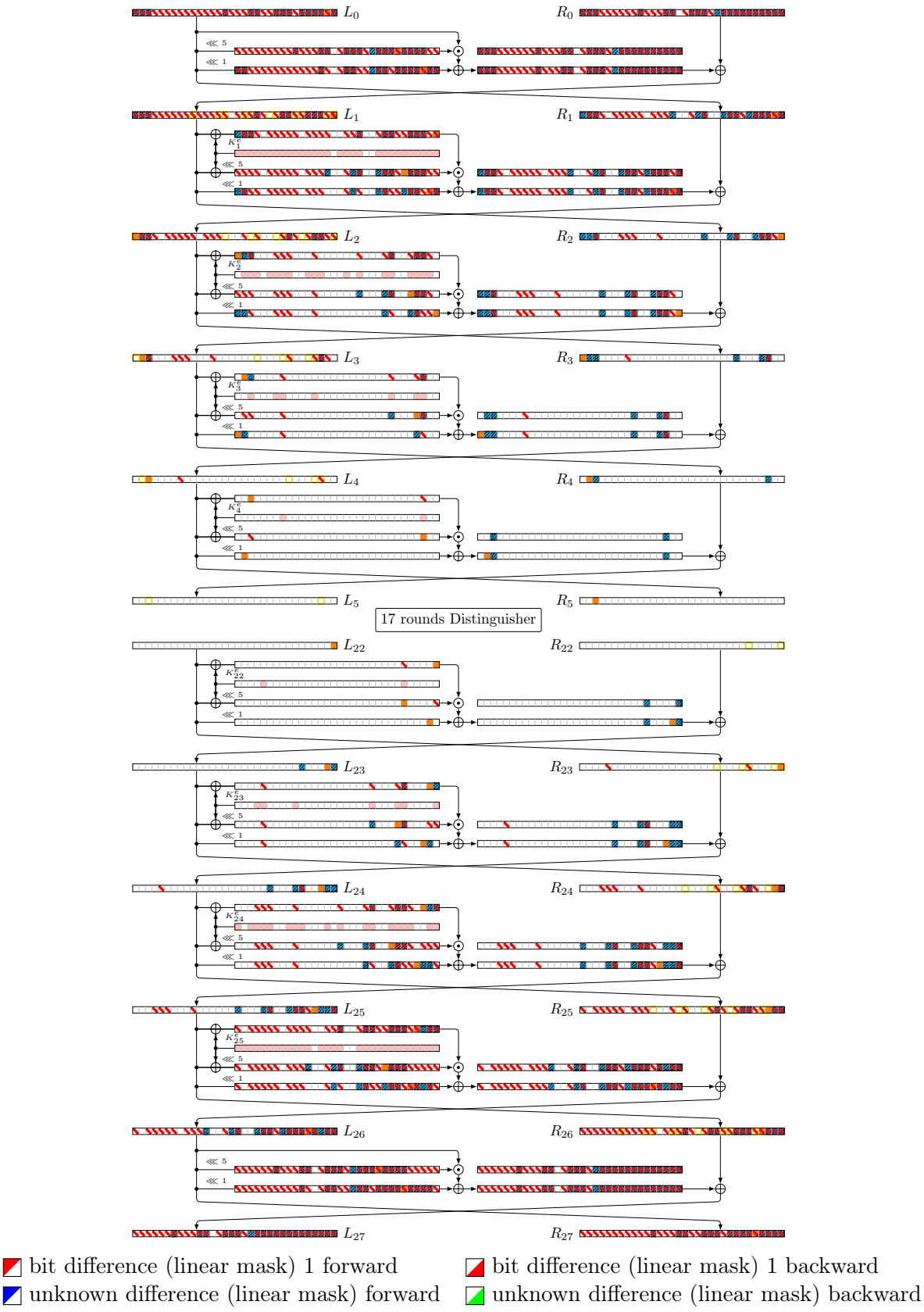


Figure 4.12: Key recovery attack on 27-round Simeck64-128.

Chapter 5

Finding Three-Subset Division Property for Ciphers with Complex Linear Layers

Division Property. Todo [54] introduced the *division property*, a groundbreaking technique for identifying integral characteristics and discovering integral distinguishers in block cipher frameworks like Feistel and SPN structures. Following this, Todo and Morii introduced the bit-based division property, known as CBDP, which is regarded as a specific instance of the division property, as noted in their 2016 paper [21]. The CBDP categorizes all vectors u in $\mathbb{F}_2^n$ into two subsets depending on whether the parity of $\bigoplus_{x\in\mathbb{X}} x^u$ is 0 or unknown, where x^u is defined as $x^u := \prod_{i=1}^{n} {x_i}^{u_i}$. Furthermore, Boura and Canteaut [132] presented a different viewpoint on the division property, the *parity set*, at CRYPTO 2016.

The complexity of using CBDP for n-bit primitives was generally around 2^n. This substantial complexity limited the broad application of CBDP. To address this issue, Xiang *et al.* [133] utilized a MILP model to search for integral distinguishers based on CBDP, applying this modeling technique to six lightweight block ciphers. By extending and enhancing this method, integral attacks have been applied to numerous ciphers, resulting in the discovery of many improved integral distinguishers [134]–[140].

Three-subset Division Property. Although CBDP can identify more precise integral distinguishers compared to other methods, its accuracy is imperfect. To achieve more accurate results, the bit-based division property using three subsets (BDPT) was introduced in [21]. BDPT classifies all vectors u in $\mathbb{F}_2^n$ into three subsets, where the parity of $\bigoplus_{x\in\mathbb{X}} x^u$ is 0, 1, or unknown. In BDPT, the *unknown subset* category from CBDP is split into a subset labelled as 1-*subset* and another subset that remains called *unknown subset*. Consequently, BDPT can more precisely identify integral characteristics than CBDP (e.g., SIMON32).

While using MILP with CBDP has proven effective, the same approach is not as successful with BDPT. With BDPT, it is necessary to monitor the propagation of the division property in two sets: $\mathbb{K}$ (the *unknown subset*) and $\mathbb{L}$ (the 1-*subset*). Additionally, the impact of $\mathbb{L}$ on $\mathbb{K}$ must be tracked due to the `Key-XOR` operation. This makes developing an automatic search algorithm based on BDPT more complex.

First, Hu *et al.* [22] introduced the variant three subset division property (VTDP) to improve some integral distinguishers. However, it does lose some accuracy compared to BDPT. To address this, Wang *et al.* [23] developed a method for modeling the propagation

of BDPT. Recently, Liu *et al.* [24] introduced the model set approach to find integral distinguishers based on BDPT. Both methods have been used on block ciphers utilizing simple bit permutations as their linear layer.

Symmetric cryptographic primitives are designed to include both non-linear and linear components to enhance security. Non-linearity, often achieved through S-boxes, plays a crucial role in providing the confusion property, which obscures the relationship between the plaintext and ciphertext. On the other hand, diffusion, which spreads the influence of each bit across the ciphertext, is typically achieved through linear operations. These operations can be simple, such as bit-permutation, or more complex, such as the use of MDS matrices, as seen in the MixColumns step of the block cipher AES. Therefore, when a block cipher is said to have a complex linear layer, it refers to the use of multiplication with an MDS matrix as its linear layer.

Motivation. According to the algorithm for modeling BDPT propagation described in [23], it can be efficiently applied to any block cipher, provided the round function can be divided into several suitable parts. However, it is challenging to model BDPT propagation for ciphers with complex linear layers. Next, Liu *et al.* [24] introduced the model set method to search for BDPT, where they created r different MILP models for r-round block ciphers, which is somewhat complicated. These methods have been used on block ciphers with simple bit permutation linear layers. Therefore, the following question arises:

Is the MILP method of BDPT propagation efficiently applicable for ciphers with complex linear layers?

Contributions of this chapter.

To answer this question, we propose a method to accurately determine BDPT propagation through complex linear layers. Then, we develop an automatic search algorithm for BDPT in this chapter. The details of our technical contributions are listed as follows:

Model the BDPT Propagation of Binary Linear Layer.

We propose a new method to precisely illustrate BDPT propagation through binary (complex) linear layers. This method helps us construct a precise MILP model of BDPT propagation for ciphers with complex (binary) linear layers. Specifically, we observe that the rows of the primitive matrix corresponding to the binary mixcolumn matrix can be grouped into cosets. These cosets have the property that rows in different cosets do not share nonzero entries in the same column. Using this property, we can accurately determine BDPT propagation and describe it using minimal inequalities. It is important to note that for any cipher whose mixcolumn matrix has this property, we can accurately determine the propagation of BDPT in those ciphers.

Construction of Automatic Search Algorithm for BDPT.

To search for BDPT, we start by creating MILP models for the round function's components in block ciphers independent of the encryption key. Incorporating a `Key-XOR` operation introduces new vectors from set $\mathbb{L}$ into set $\mathbb{K}$. Accurately modeling the `Key-XOR` operation is

a challenging task. To address this, we develop a new efficient algorithm that uses MILP techniques to precisely model each Key-XOR operation. Finally, by choosing suitable initial BDPT configurations and stopping criteria, we create an automatic search algorithm that accurately characterizes BDPT propagation using only two MILP models, which is more straightforward, and simple than the approach described in [24]. Additionally, we demonstrate the correctness of our search algorithm.

Applications.

As for the application of our methodology, first time we apply BDPT on block ciphers with complex linear layers. We use our automatic search model to search integral distinguishers of PRINCE [25], MANTIS [26], KLEIN [27], PRIDE [28], SIMON [14], and SIMON(102) [141]. The results are shown in Table 5.1.

Initially, our method is applied to PRINCE and MANTIS, both of which feature a binary linear layer. We establish a $2+2$ round integral distinguisher for PRINCE, surpassing the previous best result by one round [142]. Similarly, for MANTIS, we identify a $3+3$ round integral distinguisher, surpassing the previous best by one round [142]. In this context, a refers to rounds preceding the middle layer, b denotes rounds following the middle layer, and $a+b$ indicates the cumulative number of rounds examined.

Table 5.1: Summarization of Integral Distinguishers

Cipher	Data	Round	Constant bits	Time	Ref.
MANTIS	2^{32}	$3+2$	16	–	[142]
	$\mathbf{2^{63}}$	$\mathbf{3+3}$	**64**	**2h8m**	**This work**
PRINCE	2^{32}	$2+1$	64	–	[142]
	$\mathbf{2^{63}}$	$\mathbf{2+2}$	**64**	**21h45m**	**This work**
PRIDE*	–	8	–	–	[143]
	$\mathbf{2^{63}}$	**9**	**32**	**2h35min**	**This work**
KLEIN	2^{32}	5	64	–	[144]
	$\mathbf{2^{62}}$	**6**	**64**	**45min**	**This work**

* In [143], the authors have only mentioned that PRIDE64 has 8-round integral distinguisher and no other information is available best known to us.

To complete our BDPT analysis on ciphers with complex linear layers, we apply our method to KLEIN and PRIDE which have non-binary linear layers. As there are no known results on them related to CBDP, then we first apply MILP based CBDP on them and find 6-round and 9-round integral distinguishers for KLEIN and PRIDE respectively, which are one more rounds to previous best integral distinguishers [143], [144]. Subsequently, we apply our MILP-based BDPT method, finding integral distinguishers consistent with those from CBDP. Finally, we apply our approach to all variants of SIMON and SIMON (102) block ciphers, achieving results comparable to previous longest distinguishers [24], but in less time.

Organization.

This chapter is structured as follows: Section 5.1 explores the MILP technique for CBDP. Section 5.2 explores the MILP technique for modeling fundamental operations within the round function of a block cipher, along with precise modeling of binary linear layers in BDPT. Section 5.3 investigates the formulation of initial and stopping criteria and details the search algorithm. Section 5.4 demonstrates various applications of our model. Finally, Section 5.5 provides the concluding remarks for the chapter.

5.1 Mixed Integer Linear Programming (MILP) and Its Application to Division Property

Mixed-Integer Linear Programming (MILP) is a mathematical optimization technique used to solve problems that involve both continuous and discrete variables. In MILP, the objective is to optimize a linear function subject to linear constraints, where some variables are restricted to be integers, while others can take continuous values. MILP is widely applicable in fields such as operations research, finance, and cryptography.

In the context of symmetric cryptanalysis, MILP has become an important tool for formulating and solving cryptanalytic problems. It is particularly useful for representing complex relationships and constraints involved in attacks on block ciphers and hash functions. For example, MILP can be used to model key recovery attacks, differential attacks, and finding distinguishing features of cryptographic algorithms. By encoding the problem as a set of linear equations, MILP solvers can efficiently search for solutions that satisfy the constraints, enabling attackers to recover cryptographic keys, identify weaknesses, or break certain cryptographic schemes. In this section, we discuss how to construct MILP model in order to construct division property based integral distinguishers.

5.1.1 The MILP Model for CBDP

The authors in [133] employed the MILP method in their Asiacrypt 2016 paper to detect distinguishers using CBDP. This technique allowed them to analyze block ciphers of substantial complexity. Firstly, they unveiled the idea of CBDP trail and then the MILP models for basic operations as follows:

Definition 5.1 (CBDP Trail [133]) *Consider the propagation of the division property* $\{k^0\} \equiv \mathbb{K}_0 \xrightarrow{f_1} \mathbb{K}_1 \xrightarrow{f_2} \mathbb{K}_2 \xrightarrow{f_3} \ldots$. *Moreover, for any vector* $k^i \in \mathbb{K}_i (i \geqslant 1)$, *there must exist a vector* $k^{i-1} \in \mathbb{K}_{i-1}$ *such that* k^{i-1} *can propagate to* k^i *by CBDP propagation rules. Furthermore, for* $(k^0, k^1, \ldots, k^r) \in \mathbb{K}_0 \times \mathbb{K}_1 \times \ldots \times \mathbb{K}_r$, *if* k^{i-1} *can propagate to* k^i *for all* $i \in \{1, 2, \ldots, r\}$, *we call* $(k^0, k^1, \ldots, k^r)$ *an r-round CBDP trail.*

Proposition 5.1 (COPY [133]) *Let* $F : \mathbb{F}_2 \to \mathbb{F}_2^n$ *such that* $F(x) = (y_0, y_1, \ldots, y_{n-1})$ *where* $y_0 = y_1 = \ldots = y_{n-1} = x$, *and* $\boldsymbol{u} \to (v_0, v_1, \ldots, v_{n-1})$ *be a CBDP trail w.r.t the function* F. *Then, the valid CBDP trails satisfy*

$$\texttt{COPY}(\boldsymbol{u}, v_0, \ldots, v_{n-1}) : \boldsymbol{u} - v_0 - v_1 - \ldots - v_{n-1} = 0.$$

where u, v_i are binary variables, for all $0 \leqslant i \leqslant (n-1)$, representing the input and output CBDP w.r.t the COPY *function.*

Proposition 5.2 (XOR [133]) *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2$ such that $F(x_0, x_1, \ldots, x_{n-1}) = y$ where $y = x_0 \oplus \ldots \oplus x_{n-1}$, and $(u_0, u_1, \ldots, u_{n-1}) \to v$ be a CBDP trail w.r.t the function F. Then, the valid CBDP trails satisfy*

$$\texttt{XOR}(u_0, u_1, \ldots, u_{n-1}, v) : v - u_0 - u_1 - \ldots - u_{n-1} = 0.$$

where u_i, v are binary variables, for all $0 \leqslant i \leqslant (n-1)$, representing the input and output CBDP w.r.t the XOR *function.*

Proposition 5.3 (AND [133]) *Let $F : \mathbb{F}_2^n \to \mathbb{F}_2$ such that $F(x_0, x_1, \ldots, x_{n-1}) = y$ where $y = \bigwedge_{i=0}^{n-1} x_i$, and $(u_0, u_1, \ldots, u_{n-1}) \to v$ be a CBDP trail w.r.t the function F. Then, the valid CBDP trails satisfy*

$$\texttt{AND}(u_0, u_1, \ldots, u_{n-1}, v) : v \geqslant u_i, \text{ for all } i \in \{0, 1, n-1\}.$$

where u_i, v are binary variables, for all $0 \leqslant i \leqslant (n-1)$, representing the input and output CBDP w.r.t the AND *function.*

MILP model for S-box [133]. The rule outlined in BDPT Rule 2.11 can be used to derive the CBDP propagation property for an S-box. Subsequently, the $\texttt{inequality} - \texttt{generator}()$ function in SAGE software can be employed to obtain a set of linear inequalities.

In their work [133], the authors formulated the CBDP propagation of fundamental operations (such as COPY, XOR, AND, and S-box) using linear inequalities. This approach allowed them to construct a MILP model encompassing all potential CBDP trails starting from a given initial CBDP. By leveraging the division trail, the task of identifying the CBDP was converted into the problem of finding a division trail ending at a unit vector. For further details, refer to [133].

5.2 The MILP Model for BDPT

Suppose E_r is a r-round iterated block cipher whose round function f_i, for $i \in [r]$, consists of a non-linear layer, linear layer, and Key-XOR operation. Let f_k^i be the Key-XOR operation, and f_e^i be the rest of the operations in the i-th round function f_i i.e.

$$f_i = f_k^i \circ f_e^i$$

Let, the input multi set $\mathbb{X}$ to the block cipher E_r has initial BDPT as $D^{1^n}_{\mathbb{K}_0=\{k\},\mathbb{L}_0=\{\ell\}}$, and after propagating through i rounds, for any $i \in [r]$, we denote the output BDPT as $D^{1^n}_{\mathbb{K}_i,\mathbb{L}_i}$. Now, for the function f_e^i, we denote the BDPT propagation as

$$f_e^i(\mathbb{K}_{i-1}) = \overline{\mathbb{K}}_{i-1}, \quad f_e^i(\mathbb{L}_{i-1}) = \overline{\mathbb{L}}_{i-1}$$

We can evaluate the BDPT propagation for $\mathbb{K}$ (*unknown* subset) and $\mathbb{L}$ (1 *subset*) independently as per the BDPT propagation rules for linear and non-linear layers. Now, for the

operation f_k^i, according to BDPT Rule 2.10, some new vectors are produced from the vectors in $\overline{\mathbb{L}}_{i-1}$. These new vectors, along with the vectors in $\overline{\mathbb{K}}_{i-1}$, form the set $\mathbb{K}_i$. However, the set $\mathbb{L}_i$ is same as $\overline{\mathbb{L}}_{i-1}$.

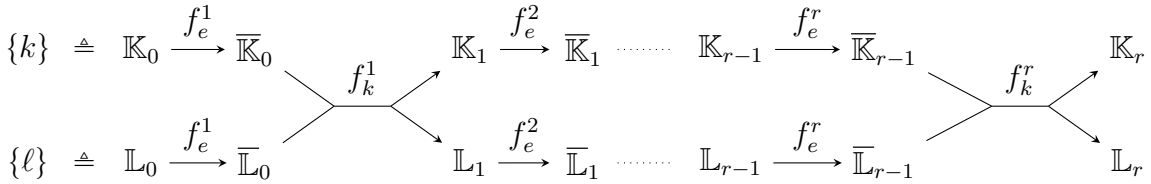
Now, we divide the operation f_k^i into two parts, say f_1^i, f_2^i such that f_1^i is the operation where new elements are produced from each element in $\overline{\mathbb{L}}_{i-1}$ according to BDPT Rule 2.10, and f_2^i is the operation which includes the new vectors and the vectors from $\overline{\mathbb{K}}_{i-1}$ in $\mathbb{K}_i$ which is as follows:

$$(\mathbb{K}_i, \mathbb{L}_i) = f_k^i(\overline{\mathbb{K}}_{i-1},\ \overline{\mathbb{L}}_{i-1}) = (f_2^i(f_1^i(\overline{\mathbb{L}}_{i-1}),\ \overline{\mathbb{K}}_{i-1}),\ \overline{\mathbb{L}}_{i-1}) \tag{5.1}$$

Precisely, f_2^i is the union operation i.e., $\mathbb{K}_i = f_1^i(\overline{\mathbb{L}}_{i-1}) \cup \overline{\mathbb{K}}_{i-1}$.

To model the propagation of BDPT for the operations f_e^i and f_k^i for all $i \in [r]$, we reintroduce a notion named BDPT ***trail***. [1]

Definition 5.2 (BDPT Trail) *Let $\mathbb{X}$ denote the input multi set to the block cipher, initialized with the BDPT $D^{1^n}_{\mathbb{K}_0=\{k\},\,\mathbb{L}_0=\{\ell\}}$. After propagating through r rounds using functions f_e^i and f_k^i, for all $i \in [r]$, the resulting BDPT is denoted as $D^{1^n}_{\mathbb{K}_r,\,\mathbb{L}_r}$, where $r \geqslant 1$. Thus, we have the following chain of BDPT propagations:*



where $\overline{\mathbb{K}}_{i-1} = f_e^i(\mathbb{K}_{i-1})$, $\overline{\mathbb{L}}_{i-1} = f_e^i(\mathbb{L}_{i-1})$, and $(\mathbb{K}_i,\ \mathbb{L}_i) = f_k^i(\overline{\mathbb{K}}_{i-1},\ \overline{\mathbb{L}}_{i-1})$, for all $1 \leqslant i \leqslant r$. Moreover, for any vector tuple $(k^i,\ \ell^i)$, $k^i \in \mathbb{K}_i$, and $\ell^i \in \mathbb{L}_i\,(i \in [r])$, there must exist $(\bar{k}^{i-1},\ \bar{\ell}^{i-1})$, where $\bar{k}^{i-1} \in \overline{\mathbb{K}}_{i-1}$, and $\bar{\ell}^{i-1} \in \overline{\mathbb{L}}_{i-1}$ such that $(\bar{k}^{i-1}, \bar{\ell}^{i-1})$ propagate to $(k^i,\ \ell^i)$ by BDPT propagation rule of `Key-XOR`, and there must exist $(k^{i-1},\ \ell^{i-1}) \in \mathbb{K}_{i-1} \times \mathbb{L}_{i-1}$ such that k^{i-1} propagate to $\bar{k}^{i-1}$, and ℓ^{i-1} propagate to $\bar{\ell}^{i-1}$ by BDPT propagation rules of linear and non-linear layers. Furthermore, for $(k^0,\ \ell^0), \ldots, (k^r,\ \ell^r) \in \mathbb{K}_0 \times \mathbb{L}_0 \times \ldots \times \mathbb{K}_r \times \mathbb{L}_r$, if $(k^{i-1},\ \ell^{i-1})$ can propagate to $(k^i,\ \ell^i)$ for all $i \in \{1, 2, \ldots, r\}$, we call

$$(k^0,\ \ell^0) \xrightarrow{f_e^1,\, f_k^1} (k^1,\ \ell^1) \xrightarrow{f_e^2,\, f_k^2} \ldots \xrightarrow{f_e^r,\, f_k^r} (k^r,\ \ell^r)$$

an r-round BDPT trail.

Now, to model BDPT trail, we propose Proposition 5.4 according to Definition 5.2.

Proposition 5.4 *Consider an input multi set $\mathbb{X}$ with an initial BDPT denoted by $D^{1^n}_{\{k\},\{\ell\}}$, and let $D^{1^n}_{\mathbb{K}_r,\mathbb{L}_r}$ represent the BDPT of the output multi set after r rounds of propagation. Then, the set of initial components of the final vectors in all r-round BDPT trails starting with vector (k, ℓ) is identical to $\mathbb{K}_r$. The set of second components of these final vectors is identical to $\mathbb{L}_r$.*

Proof of this Proposition 5.4 directly follows from Definition 5.2.

[1] In [24], the authors have defined BDPT ***trail***. We rewrite it according to our notations.

5.2.1 Some Observations on BDPT Propagation Rule for S-box

The S-box is crucial in block ciphers, often serving as the sole nonlinear component in many designs. Inspired by the algorithm of calculating CBDP trails of S-box [133], the authors in [23] introduced a procedure to costruct BDPT division trails of S-box, and we have mentioned the rule in BDPT Rule 2.11. Let, the initial BDPT of S-box is $D^{1^n}_{\mathbb{K},\,\mathbb{L}=\{\ell\}}$, and according to the BDPT Rule 2.11, we have found the sets $\underline{\mathbb{K}}$, and $\underline{\mathbb{L}}$ from $\mathbb{K}$ and $\mathbb{L}$ respectively as follows:

$$\begin{cases} \underline{\mathbb{K}} = \{u' \in \mathbb{F}_2^n \ : \ \text{for any } u \in \mathbb{K}, \ \text{if } y^{u'} \text{ includes any term } x^v \text{ satisfying } v \succeq u\} \\ \underline{\mathbb{L}} = \{u \in \mathbb{F}_2^n \ : \ y^u \text{ contains the term } x^\ell\} \end{cases} \tag{5.2}$$

Now, according to the BDPT Rule 2.11, the output BDPT would be $D^{1^n}_{\mathbb{K}',\,\mathbb{L}'}$ which is as follows:

$$\mathbb{K}' = \textbf{\textit{Reduce}}\mathbf{0}(\underline{\mathbb{K}}),\ \mathbb{L}' = \textbf{\textit{Reduce}}\mathbf{1}(\underline{\mathbb{K}}, \ \underline{\mathbb{L}})$$

Therefore, it is obvious that, $\mathbb{K}' \subseteq \underline{\mathbb{K}}$, and $\mathbb{L}' \subseteq \underline{\mathbb{L}}$. Here, we come to two observations as follows:

Observation 1 $\mathbb{L}'$ *does not contain* $\mathbf{1}$ *vector.*

According to the BDPT propagation rule of S-box, as $\mathbb{L}' = \textbf{\textit{Reduce}}\mathbf{1}(\underline{\mathbb{K}}, \ \underline{\mathbb{L}})$, and for any $u \in \underline{\mathbb{K}}$, $\mathbf{1} \succeq u$, then $\mathbb{L}'$ does not contain the $\mathbf{1}$ vector.

Observation 2 *If* $\mathbb{L} = \{\mathbf{0}\}$, *then* $\mathbb{L}' = \{\mathbf{0}\}$.

Whenever, $\mathbb{L} = \{\mathbf{0}\}$, then $\bigoplus_{x\in\mathbb{X}} x^{\mathbf{0}} = 1$ which implies that the input multiset $\mathbb{X}$ contains a constant term. Therefore, for all $u > \mathbf{0}$, $\bigoplus_{x\in\mathbb{X}} x^u =$ unknown. Hence, trivially $\bigoplus_{y\in\mathbb{Y}} y^{\mathbf{0}} = 1$ and $\mathbb{L}' = \{\mathbf{0}\}$ where $\mathbb{Y}$ is the output multi set.

Therefore, lets assume, given an n-bit S-box and its input BDPT $D^{1^n}_{\mathbb{K}=\{k\},\,\mathbb{L}=\{\ell\}}$, BDPT Rule 2.11 returns the output BDPT $D^{1^n}_{\mathbb{K}',\,\mathbb{L}'}$. Hence, for any vector $k' \in \mathbb{K}'$, the pair (k, k') constitutes a valid division trail for $\mathbb{K}'$ of the S-box. This principle similarly applies to $\mathbb{L}'$. Given that the vector ℓ does not influence the propagation of the vector k through the S-box, a comprehensive list of division trails for $\mathbb{K}'$ can be obtained by iterating over all $k \in \mathbb{F}_2^n$ [133].

Similarly, for a particular input vector $\ell \in \mathbb{F}_2^n$, we will obtain a particular set of division trails for $\underline{\mathbb{L}}$ using Equation 5.2 and then using Observation 1, and Observation 2 we will remove some invalid division trails from $\underline{\mathbb{L}}$ and obtain a set of division trail for $\mathbb{L}'$. Therefore, if we try all the 2^n possible input vector ℓ, we will get a complete list of division trails for $\mathbb{L}'$.

In [23], the authors included some invalid BDPT trail for $\mathbb{L}'$ set while obtaining a complete list of division trails for $\mathbb{L}'$. In [24], the authors removed those invalid BDPT trails from $\mathbb{L}'$ by introducing another algorithm which is a bit complicated. Now, our approach is equivalent to the idea of removing invalid trails, described in [24] in a much simplified manner using two aforementioned observations, constructed from BDPT Rule 2.11).

In Table 5.2, we present the complete lists of all the division trails for $\mathbb{L}$ of PRINCE S-box according to our method. We observe that it is the same division trails for $\mathbb{L}$ of PRINCE, if we apply the method the authors described in [24]. Therefore, after getting the BDPT trails for $\mathbb{K}$ and $\mathbb{L}$ of S-box, we construct the linear inequalities using the method described in [133] whose feasible solutions are exactly those BDPT trails which are shown as follows:

Table 5.2: Division Trails for $\mathbb{L}$ of PRINCE S-box

Input ℓ	Output $\mathbb{L}$
$[0,0,0,0]$	$[0,0,0,0]$
$[0,0,0,1]$	$[0,1,0,0], [0,1,0,1], [0,1,1,0], [0,1,1,1], [1,1,0,0], [1,1,0,1], [1,1,1,0]$
$[0,0,1,0]$	$[1,0,0,0], [1,0,0,1], [1,0,1,0], [1,0,1,1]$
$[0,0,1,1]$	$[0,0,0,1], [0,0,1,1], [0,1,0,0], [0,1,0,1], [0,1,1,0], [0,1,1,1]$ $[1,1,0,0], [1,1,0,1], [1,1,1,0]$
$[0,1,0,0]$	$[0,0,0,1], [0,0,1,1], [1,0,0,1], [1,0,1,1]$
$[0,1,0,1]$	$[0,0,1,0], [0,1,0,1], [0,1,1,0], [0,1,1,1], [1,0,1,0], [1,1,0,1], [1,1,1,0]$
$[0,1,1,0]$	$[0,0,0,1], [0,0,1,0], [1,0,0,0], [1,0,1,1]$
$[0,1,1,1]$	$[0,0,0,1], [0,0,1,0], [0,0,1,1], [0,1,0,1], [0,1,1,0], [0,1,1,1], [1,0,0,0]$ $[1,0,0,1], [1,0,1,0], [1,1,0,1], [1,1,1,0]$
$[1,0,0,0]$	$[0,0,0,1], [0,0,1,1], [0,1,0,0], [0,1,1,0], [1,0,0,0], [1,0,0,1]$ $[1,0,1,0], [1,0,1,1]$
$[1,0,0,1]$	$[0,0,0,1], [0,0,1,1], [0,1,0,0], [0,1,1,0], [1,1,0,0], [1,1,0,1], [1,1,1,0]$
$[1,0,1,0]$	$[0,0,1,0], [0,1,0,0], [0,1,1,0], [1,0,0,1], [1,0,1,0], [1,0,1,1]$
$[1,0,1,1]$	$[0,1,0,0], [0,1,1,0], [1,0,0,0], [1,1,0,0], [1,1,0,1], [1,1,1,0]$
$[1,1,0,0]$	$[0,0,0,1], [0,0,1,1], [1,0,0,0], [1,0,0,1], [1,0,1,0], [1,0,1,1], [1,1,0,0]$
$[1,1,0,1]$	$[0,0,1,1], [0,1,0,1], [1,0,0,0], [1,1,0,0], [1,1,0,1],$
$[1,1,1,0]$	$[0,0,1,0], [0,1,0,0], [0,1,0,1], [1,0,0,1], [1,0,1,0]$ $[1,0,1,1], [1,1,0,1], [1,1,1,0]$
$[1,1,1,1]$	$[1,1,1,1]$

Linear Inequalities description BDPT of PRINCE S-box

Below are the inequalities that define the PRINCE S-box, representing the feasible solutions corresponding to the division trails for $\mathbb{K}$ of the PRINCE S-box where $(x_3, x_2, x_1, x_0) \rightarrow (y_3, y_2, y_1, y_0)$ denotes a division trail.

$$\begin{cases} x_3 + x_2 + x_1 + 4x_0 - 2y_3 - 2y_3 - 2y_3 - 2y_3 \geqslant -1 \\ 3x_3 - y_3 - y_2 - y_1 - y_0 \geqslant -1 \\ -2x_3 - x_2 - x_1 - 2x_0 + 5y_3 + 5y_2 + 4y_1 + 4y_0 \geqslant 0 \\ 3x_2 - y_3 - y_2 - y_1 - y_0 \geqslant -1 \\ -5x_3 - 5x_2 - 5x_1 - 4x_0 + y_3 + 2y_2 + 2y_1 + y_0 \geqslant -13 \\ -y_3 - y_2 - y_1 + 2y_0 \geqslant -1 \\ x_1 - y_3 - y_0 \geqslant -1 \\ -x_3 - x_1 + y_3 - y_0 \geqslant -2 \\ -x_2 - x_0 + 2y_3 + y_2 + 2y_1 + 2y_0 \geqslant 0 \\ -x_3 - x_0 + y_3 + y_2 + y_0 \geqslant -1 \\ -2x_3 - 2x_2 - 2x_0 + y_3 - y_2 - y_1 + 2y_0 \geqslant -5 \end{cases} \tag{5.3}$$

Here are the inequalities that define the PRINCE S-box, outlining the feasible solutions corresponding to the division trails for $\mathbb{L}$ within the PRINCE S-box where $(x_3, x_2, x_1, x_0) \rightarrow (y_3, y_2, y_1, y_0)$ denotes a division trail.

$$\begin{cases} x_3 + x_2 + x_1 + 3x_0 - 2y_3 - 4y_2 - 2y_1 - 2y_0 \geqslant -5 \\ -2x_3 - x_2 - x_1 - 2x_0 + 5y_3 + 5y_2 + 4y_1 + 4y_0 \geqslant 0 \\ -2x_3 + x_2 + x_1 - 6x_0 - 6y_3 + 4y_2 - 2y_1 - 3y_0 \geqslant -13 \\ 4x_3 - x_2 + 3x_1 + 6x_0 - y_3 - 2y_2 - 2y_1 + 4y_0 \geqslant 0 \\ -3x_3 + x_2 - 3x_1 - x_0 + 2y_3 - y_1 - 3y_0 \geqslant -8 \\ 2x_3 + x_2 + 2x_1 - x_0 + 2y_2 - y_0 \geqslant 0 \\ -2x_3 - 2x_2 + x_1 + y_3 - y_2 - y_1 + y_0 \geqslant -4 \\ x_3 + x_2 + 2x_0 + y_3 - y_2 - y_0 \geqslant 0 \\ x_3 + 5x_2 + x_1 - 3x_0 + y_3 + 5y_2 - 2y_1 + 4y_0 \geqslant 0 \\ x_3 - 2x_2 + x_1 + y_3 + 2y_1 + 2y_0 \geqslant 0 \\ -x_3 - 2x_2 - 2x_1 - x_0 - y_3 - y_2 + y_1 + y_0 \geqslant -6 \\ x_2 + x_0 - y_3 - y_2 \geqslant -1 \\ -x_3 - x_1 + x_0 + y_3 + 2y_2 + 2y_1 + y_0 \geqslant 0 \\ -x_3 + x_2 + y_3 - y_2 - y_0 \geqslant -2 \\ -x_3 - x_2 + x_1 + x_0 - 2y_2 - y_1 - y_0 \geqslant -4 \\ -x_1 + x_0 + y_3 - y_1 - y_0 \geqslant -2 \\ x_3 - x_2 - x_1 + x_0 - y_3 + y_1 - y_0 \geqslant -3 \\ x_3 + x_2 + x_1 + 3x_0 - 3y_3 - 5y_2 - 3y_1 - 3y_0 \geqslant -8 \end{cases} \tag{5.4}$$

5.2.2 MILP Model of BDPT for Complex Linear Layer

In this section, we establish the idea to construct the MILP model of BDPT for a complex linear layer represented by a matrix $M = (m_{i,j})_{s\times s} \in \mathbb{F}_{2^m}^{s\times s}$. Given the irreducible polynomial of the field $\mathbb{F}_2^m$, the representation of the matrix over $\mathbb{F}_2$ is unique, which we call the primitive matrix of M and is denoted by $M' = (m'_{i,j})_{n\times n}$ where $m'_{i,j} \in \mathbb{F}_2$ and $n = m \times s$. Therefore, if each $m_{i,j}$ in M which is a polynomial in the extension field $F_{2^m} \simeq \mathbb{F}[x]/(f)$, where f is the irreducible polynomial over $\mathbb{F}_2$ with degree m, is either 0 or 1, then M is called *binary matrix* and otherwise M is *non-binary matrix*.

Therefore, block ciphers with complex linear layers can be partitioned into two parts: (i) block ciphers with binary linear layer and (ii) block ciphers with non-binary linear layer, depending on the binary or non-binary matrix as its linear layer. Examples of block ciphers having binary linear layer are MIDORI, SKINNY, CRAFT, MANTIS etc., and AES, LED, KLEIN, PRIDE etc. have non-binary linear layer.

Now, an obvious way to model the BDPT propagation through any complex linear layer, i.e., $u_1 \xrightarrow{M} v_1$ in $\mathbb{K}$ subset, and $u_2 \xrightarrow{M} v_2$ in $\mathbb{L}$ subset is that one can introduce some auxiliary binary variables and decompose it into the COPY and XOR operations. Consequently, the BDPT propagation through the linear layer can be modeled by adhering to the BDPT propagation rules for COPY and XOR. The apparent advantage of this model is that using this technique, we can model BDPT propagation of any complex linear layer.

In [136], [137], the authors have shown that using this technique, one may introduce many invalid division trails in $\mathbb{K}$ subset. Now, here we establish that if we use this COPY-XOR technique to handle binary linear layer, then many invalid division trails may be added to the $\mathbb{L}$ subset as well.

An Example of Binary Matrices : The COPY-XOR Technique Cannot Find Accurate BDPT.

We provide an example of binary matrices where the COPY-XOR technique fails to trace its BDPT accurately.

Example 5 *Assume the linear layer forms a matrix*

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \in \mathbb{F}_2^{4\times 4}$$

Let, $x = (x_4, x_3, x_2, x_1)^T$ represent the input and $y = (y_4, y_3, y_2, y_1)^T$ the output of matrix M i.e., $y = Mx$. This multiplication is then expressed in a vectorial Boolean form as follows:

$$\begin{cases} y_4 = x_1 + x_2 + x_3 \\ y_3 = x_1 + x_2 + x_4 \\ y_2 = x_1 + x_3 + x_4 \\ y_1 = x_2 + x_3 + x_4 \end{cases}$$

Let, $\boldsymbol{x} = (\boldsymbol{x}_4, \boldsymbol{x}_3, \boldsymbol{x}_2, \boldsymbol{x}_1)$*, and* $\boldsymbol{y} = (\boldsymbol{y}_4, \boldsymbol{y}_3, \boldsymbol{y}_2, \boldsymbol{y}_1)$ *be the input and output division property corresponding to* $\mathbb{L}$ *subset. Note that,* x_1 *appears in the first, second and third equations, so according to the BDPT propagation rule of* `COPY` *(BDPT Rule 2.7), we need to apply* `COPY` *operation to* $\boldsymbol{x}_1$*. We introduce three binary variables* $\boldsymbol{x}_1^1$*,* $\boldsymbol{x}_1^2$*, and* $\boldsymbol{x}_1^3$ *to represent its output, i.e.,*

$$\boldsymbol{x}_1 \xrightarrow{\texttt{COPY}} (\boldsymbol{x}_1^1, \boldsymbol{x}_1^2, \boldsymbol{x}_1^3).$$

Similarly, $(\boldsymbol{x}_2^1, \boldsymbol{x}_2^2, \boldsymbol{x}_2^3)$*,* $(\boldsymbol{x}_3^1, \boldsymbol{x}_3^2, \boldsymbol{x}_3^3)$*, and* $(\boldsymbol{x}_4^1, \boldsymbol{x}_4^2, \boldsymbol{x}_4^3)$ *are introduced to represent the* `COPY` *of* x_2, x_3*, and* x_4*, respectively. Therefore, we model* `XOR` *operations with the help of the binary variables*

$$\begin{cases} (\boldsymbol{x}_1^1, \boldsymbol{x}_2^1, \boldsymbol{x}_3^1) \xrightarrow{\texttt{XOR}} \boldsymbol{y}_4 \\ (\boldsymbol{x}_1^2, \boldsymbol{x}_2^2, \boldsymbol{x}_4^1) \xrightarrow{\texttt{XOR}} \boldsymbol{y}_3 \\ (\boldsymbol{x}_1^3, \boldsymbol{x}_3^2, \boldsymbol{x}_4^2) \xrightarrow{\texttt{XOR}} \boldsymbol{y}_2 \\ (\boldsymbol{x}_2^3, \boldsymbol{x}_3^3, \boldsymbol{x}_4^3) \xrightarrow{\texttt{XOR}} \boldsymbol{y}_1 \end{cases}$$

Now, if we consider a candidate division trail $\boldsymbol{x} = (0,0,1,1) \xrightarrow{M} \boldsymbol{y} = (1,1,0,0)$*, then a set of assignments the auxiliary binary variables satisfy as follows:*

$$\begin{cases} (\boldsymbol{x}_1^1, \boldsymbol{x}_2^1, \boldsymbol{x}_3^1) = (0,0,0) \\ (\boldsymbol{x}_1^2, \boldsymbol{x}_2^2, \boldsymbol{x}_4^1) = (0,0,0) \\ (\boldsymbol{x}_1^3, \boldsymbol{x}_3^2, \boldsymbol{x}_4^2) = (1,0,0) \\ (\boldsymbol{x}_2^3, \boldsymbol{x}_3^3, \boldsymbol{x}_4^3) = (1,0,0) \end{cases}$$

Then, from the `COPY-XOR` *method, we know* $\boldsymbol{x} \xrightarrow{M} \boldsymbol{y}$ *should be a valid division trail in* $\mathbb{L}$*. But,*

$$y^{(1,1,0,0)} = x_1 \oplus x_1x_4 \oplus x_1x_2 \oplus x_1x_2 \oplus x_2 \oplus x_2x_4 \oplus x_1x_3 \oplus x_2x_3 \oplus x_4x_3,$$

thus both of the x_1x_2 *terms cancel each other. Therefore,* $y^{(1,1,0,0)}$ *does not contain the term* $x^{(0,0,1,1)}$*. From the definition of* BDPT*, it means* $(0,0,1,1) \xrightarrow{M} (1,1,0,0)$ *is invalid.*

Table 5.3 shows the division trail for $\mathbb{L}$ *subset of linear transformation* M*. In Table 5.3, the bold vectors are invalid division trails for* $\mathbb{L}$ *of linear transformation* M*, which are produced following the* `COPY-XOR` *technique.*

Exact BDPT Modelization for Ciphers having Binary Linear Layer.

Given a binary matrix $M = (m_{i,j})_{s\times s} \in \mathbb{F}_{2^m}^{s\times s}$, and denote $n = m \times s$, we can derive an equivalent matrix working at a bit level which is called primitive matrix $M' = (m'_{i,j})_{n\times n} \in \mathbb{F}_2^{n\times n}$. Now, M' has $n = ms$ number of rows, which we denote as $R_0, R_1, \ldots, R_{n-1}$, and define a set of all rows $\mathcal{R} = \{R_i \ : \ 0 \leqslant i \leqslant n-1\}$. Therefore, we can construct m disjoint sets $\mathcal{R}_0,\ \mathcal{R}_1, \ldots, \mathcal{R}_{m-1}$ in the following way:

$$\mathcal{R}_i = \{R_{mj+i} \ : \ 0 \leqslant j \leqslant s-1\} \text{ for all } 0 \leqslant i \leqslant m-1 \tag{5.5}$$

Now, it is obvious that $\sqcup_{i=0}^{m-1}\mathcal{R}_i = \mathcal{R}$, and $\mathcal{R}_i$ contains exactly s number of rows from M', where $0 \leqslant i \leqslant m-1$. Here, we encounter an important property: the rows in different sets

Table 5.3: Division Trails for $\mathbb{L}$ of Linear Transformation M

Input ℓ	Output $\mathbb{L}$
$[0,0,0,0]$	$[0,0,0,0]$
$[0,0,0,1]$	$[0,0,1,0], [0,1,0,0], [0,1,1,0], [1,0,0,0], [1,0,1,0], [1,1,0,0], [1,1,1,0]$
$[0,0,1,0]$	$[0,0,0,1], [0,1,0,0], [0,1,0,1], [1,0,0,0], [1,0,0,1], [1,1,0,0], [1,1,0,1]$
$[0,0,1,1]$	$[0,0,1,1], [0,1,0,1], [0,1,1,0], [1,0,0,1], [1,0,1,0], [1,1,0,1], [1,1,1,0]$ $\mathbf{[1,1,0,0], [1,0,1,1], [0,1,1,1], [1,1,1,1]}$
$[0,1,0,0]$	$[0,0,0,1], [0,0,1,0], [0,0,1,1], [1,0,0,0], [1,0,0,1], [1,0,1,0], [1,0,1,1]$
$[0,1,0,1]$	$[0,0,1,1], [0,1,0,1], [0,1,1,0, [1,0,0,1], [1,0,1,1], [1,1,0,0], [1,1,1,0]$ $\mathbf{[0,1,1,1], [1,1,0,1], [1,0,1,0], [1,1,1,1]}$
$[0,1,1,0]$	$[0,0,1,1], [0,1,0,1], [0,1,1,0, [1,0,1,0], [1,0,1,1], [1,1,0,0], [1,1,0,1]$ $\mathbf{[1,0,0,1], [1,1,1,0], [0,1,1,1], [1,1,1,1]}$
$[0,1,1,1]$	$[1,0,1,1], [1,1,0,1], [1,1,1,0], \mathbf{[0,1,1,1], [1,1,1,1]}$
$[1,0,0,0]$	$[0,0,0,1], [0,0,1,0], [0,0,1,1], [0,1,0,0], [0,1,0,1], [0,1,1,0], [0,1,1,1]$
$[1,0,0,1]$	$[0,0,1,1], [0,1,0,1], [0,1,1,1], [1,0,0,1], [1,0,1,0], [1,1,0,0], [1,1,1,0]$ $\mathbf{[0,1,1,0], [1,0,1,1], [1,1,0,1], [1,1,1,0]}$
$[1,0,1,0]$	$[0,0,1,1], [0,1,1,0], [0,1,1,1][1,0,0,1], [1,0,1,0], [1,1,0,0], [1,1,0,1]$ $\mathbf{[0,1,0,1], [1,0,1,1], [1,1,1,0], [1,1,1,1]}$
$[1,0,1,1]$	$[0,1,1,1], [1,1,0,1], [1,1,1,0]$ $\mathbf{[1,0,1,1], [1,1,1,1]}$
$[1,1,0,0]$	$[0,1,0,1], [0,1,1,0], [0,1,1,1], [1,0,0,1], [1,0,1,0], [1,0,1,1], [1,1,0,0]$ $\mathbf{[0,0,1,1], [1,1,1,0], [1,1,0,1], [1,1,1,1]}$
$[1,1,0,1]$	$[0,1,1,1], [1,0,1,1], [1,1,1,0]$ $\mathbf{[1,1,0,1], [1,1,1,1]}$
$[1,1,1,0]$	$[0,1,1,1], [1,0,1,1], [1,1,0,1]$ $\mathbf{[1,1,1,0], [1,1,1,1]}$
$[1,1,1,1]$	$[1,1,1,1]$

have no common nonzero entries in the same column, which is the key feature of a binary matrix. Exploiting this property of a binary matrix, the binary linear layer can be seen as the application of m many s-bit S-box with algebraic degree 1 in parallel.

Therefore, if $x = (x_0, x_1, \ldots, x_{n-1})$, and $y = (y_0, y_1, \ldots, y_{n-1})$ are corresponding input and output variables w.r.t the linear layer i.e., $y = M' \cdot x$, then we can write ANF of m many s-bit S-box with algebraic degree 1 as follows:

$$\begin{cases} S_0(\boldsymbol{x_0}) &= (R_0^0 \cdot \boldsymbol{x_0},\ R_m^0 \cdot \boldsymbol{x_0}, \ldots, R_{(s-1)m}^0 \cdot \boldsymbol{x_0}) \\ S_1(\boldsymbol{x_1}) &= (R_1^1 \cdot \boldsymbol{x_1},\ R_{m+1}^1 \cdot \boldsymbol{x_1}, \ldots, R_{(s-1)m+1}^1 \cdot \boldsymbol{x_1}) \\ \vdots & \\ S_{m-1}(\boldsymbol{x_{m-1}}) &= (R_{m-1}^{m-1} \cdot \boldsymbol{x_{m-1}},\ R_{2m-1}^{m-1} \cdot \boldsymbol{x_{m-1}}, \ldots, R_{sm-1}^{m-1} \cdot \boldsymbol{x_{m-1}}) \end{cases}$$

where R_{mj+i}^i is a vector which belongs to the set $\mathbb{F}_2^s$ such that $R_{mj+i}^i = (m'_{mj+i,i},\ m'_{mj+i,m+i}, \ldots, m'_{mj+i,(s-1)m+i})$, and $\boldsymbol{x_i} = (x_i,\ x_{m+i}, \ldots, x_{(s-1)m+i}) \in \mathbb{F}_2^s$ where $i = 0, 1, \ldots, m-1$, and $j = 0, 1, \ldots, s-1$.

An Example of Exact BDPT Modelization of Binary Matrix

The mixcolumn matrix M of the block cipher MANTIS can be written as follows:

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \in \mathbb{F}_{2^4}^{4\times 4}$$

Therefore, for this example, $s = 4$, and $m = 4$, and the primitive matrix M' corresponding to the matrix M is a 16×16 matrix where each matrix element is either 0 or 1, i.e., the primitive matrix $M' \in \mathbb{F}_2^{16\times 16}$ is illustrated in Equation 5.6.

$$M' = \begin{pmatrix} 0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1 \\ 1&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0 \\ 0&1&0&0&0&0&0&0&0&1&0&0&0&1&0&0 \\ 0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&0 \\ 0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1 \\ 1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0 \\ 0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0 \\ 0&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0 \\ 0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&1 \\ 1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0 \end{pmatrix} \in \mathbb{F}_2^{16\times 16} \tag{5.6}$$

Now, we can easily conclude that applying the matrix M' to a vector $x = (x_0, x_1, \ldots, x_{15}) \in \mathbb{F}_2^{16}$ is actually equivalent to performing the following 4-bit S-box in parallel:

$$S_i(x_i, x_{i+4}, x_{i+8}, x_{i+12}) = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_i \\ x_{i+4} \\ x_{i+8} \\ x_{i+12} \end{pmatrix},\ i \in \{0, 1, 2, 3\}$$

Table 5.4: Trails Corresponding to the Function f_1^i

$(\ell_0, \ell_1, \ell_2, \ell_3)$	$(\ell'_0, \ell'_1, \ell'_2, \ell'_3)$
$(0, 0, \ell_2, \ell_3)$	$(0, 1, \ell_2, \ell_3)$, $(1, 0, \ell_2, \ell_3)$, $(1, 1, \ell_2, \ell_3)$
$(0, 1, \ell_2, \ell_3)$	$(1, 1, \ell_2, \ell_3)$
$(1, 0, \ell_2, \ell_3)$	$(1, 1, \ell_2, \ell_3)$
$(1, 1, \ell_2, \ell_3)$	X

Therefore, we can construct exact BDPT trails for $\mathbb{K}$ and $\mathbb{L}$ for the mixcolumn operation and the linear inequalities whose feasible solutions are exactly those BDPT trail.

We already discuss the exact BDPT modelization of the S-box in the previous section. Applying that approach, we can get the precise BDPT trail through the binary linear layer, and then we can easily represent the BDPT trails of the binary linear layer as linear inequalities following the approach mentioned in [133]. Hence, a method is proposed to derive a set of inequalities precisely representing the valid BDPT propagations across a binary linear layer. For the ciphers with non-binary linear layer, we decompose its linear layer through the `COPY` and `XOR` operation trivially and then create a set of linear inequalities to represent the propagations within the linear layer.

5.2.3 MILP Model of BDPT for `Key-XOR`

In this section, we explain how to construct the MILP model of BDPT for the `Key-XOR` operation. As per the notation discussed above, E_r is the r-round block cipher where we denote f_i the i-th round function and f_k^i the i-th round `Key-XOR` operation. Moreover, we denote the initial and output BDPT for the `Key-XOR` operation as $(\overline{\mathbb{K}}_{i-1}, \overline{\mathbb{L}}_{i-1})$, and $(\mathbb{K}_i, \mathbb{L}_i)$, respectively. Therefore, as per BDPT Rule 2.10, we decompose f_k^i into two operations, say f_1^i, which actually produces some new elements from each elements of $\overline{\mathbb{L}}_{i-1}$ and f_2^i which includes the new vectors and the vectors from $\overline{\mathbb{K}}_{i-1}$ in $\mathbb{K}_i$, which is described in Equation 5.1. Hence, we model the operations f_1^i and f_2^i, which jointly present the MILP model for `Key-XOR` operation.

Modeling f_1^i.

In several ciphers, the round key is exclusively combined using `XOR` operation with a block segment. For simplicity, we assume this involves the leftmost s bits of the block, where s ranges from 0 to $n-1$. Let, $\overline{\mathbb{L}}_{i-1} \subseteq \mathbb{F}_2^4$ and $s = 2$ i.e., round key is `XOR`ed with the leftmost 2 bits. Therefore, according to the BDPT Rule 2.10, the function f_1^i generates $\ell' = (\ell'_0, \ell'_1, \ell'_2, \ell'_3)$ from $\ell = (\ell_0, \ell_1, \ell_2, \ell_3)$, where for every vector $\ell \in \overline{\mathbb{L}}_{i-1}$ that satisfies $\ell_i = 0$, $\ell'_i = \ell_i \vee 1$ for $i \in \{0, 1\}$, and $\ell'_j = \ell_j$ for all $j = 2, 3$. Now, we write the propagation table (Table 5.4) corresponding to the function f_1^i using which we construct linear inequalities whose feasible solutions are exactly those trails. Let, $\mathtt{l}$, and $\mathtt{l}'$ be the MILP variables corresponding to input and output BDPT corresponding to the function f_1^i, respectively. Hence, we are ready

to give linear inequalities description of these trails listed in Table 5.4 as follows:

$$\begin{cases} \mathtt{l}'_j & \geqslant & \mathtt{l}_j, & \text{for } j = 0, 1 \\ \mathtt{l}'_j & = & \mathtt{l}_j, & \text{for } j = 2, 3 \\ 2\sum_{j=0}^{1} \mathtt{l}'_j - \sum_{j=0}^{1} \mathtt{l}_j & \geqslant & 2 & \\ \sum_{j=0}^{3} \mathtt{l}'_j - \sum_{j=0}^{3} \mathtt{l}_j & \geqslant & 1 & \end{cases} \tag{5.7}$$

where $\mathtt{l}'_0$, $\mathtt{l}'_1$, $\mathtt{l}'_2$, $\mathtt{l}'_3$, $\mathtt{l}_0$, $\mathtt{l}_1$, $\mathtt{l}_2$, $\mathtt{l}_3$ are binaries.

All feasible solutions of the inequalities in Equation 5.7 are exactly the trails of the f_1^i function described above in Table 5.4, where ℓ is the input BDPT, and ℓ' is the output BDPT corresponding to the function f_1^i. Similarly, for a n-bit block cipher where $\overline{\mathbb{L}}_{i-1} \subseteq \mathbb{F}_2^n$, and the round key is XORed with the leftmost s ($0 \leqslant s \leqslant n-1$) bits, the linear inequalities we get which describe the trails $\ell \xrightarrow{f_1^i} \ell'$ (where $\mathtt{l}$, and $\mathtt{l}'$ are the MILP variables corresponding to ℓ, and ℓ', respectively) as follows:

$$\begin{cases} \mathtt{l}'_j & \geqslant & \mathtt{l}_j, & \text{for } j = 0,\ 1,\ ...,\ s-1 \\ \mathtt{l}'_j & = & \mathtt{l}_j, & \text{for } j = s,\ s+1\ \ldots, n-1 \\ s\sum_{j=0}^{s-1} \mathtt{l}'_j - (s-1)\sum_{j=0}^{s-1} \mathtt{l}_j & \geqslant & s & \\ \sum_{j=0}^{n-1} \mathtt{l}'_j - \sum_{j=1}^{n} \mathtt{l}_j & \geqslant & 1 & \end{cases} \tag{5.8}$$

where $\mathtt{l}'_0,\ \mathtt{l}'_1,\ \ldots,\ \mathtt{l}'_{n-1}, \mathtt{l}_0,\ \mathtt{l}_1,\ \ldots,\ \mathtt{l}_{n-1}$ are binaries.

Modeling f_2^i.

Once f_1^i has been applied to every element in the collection $\overline{\mathbb{L}}_{i-1}$, the set $\mathbb{L}'_{i-1}$ is obtained as follows:

$$\mathbb{L}'_{i-1} = \{\ell' \in \mathbb{F}_2^n \ :\ f_1^i(\ell) = \ell',\ \forall\ \ell \in \overline{\mathbb{L}}_{i-1}\}$$

Now, from BDPT Rule 2.10 we know that:

$$f_2^i(\overline{\mathbb{K}}_{i-1},\ \mathbb{L}'_{i-1}) = \overline{\mathbb{K}}_{i-1} \cup \mathbb{L}'_{i-1} = \mathbb{K}_i$$

Therefore, to model f_2^i, we define another function $g : (\mathbb{F}_2^2 \backslash \{(0,0),(1,1)\}) \times \overline{\mathbb{K}}_{i-1} \times \mathbb{L}'_{i-1} \to \mathbb{K}_i$ such that:

$$g(\lambda_0, \lambda_1, \bar{k}, \ell') = (\lambda_0 \wedge \bar{k}_0, \ldots, \lambda_0 \wedge \bar{k}_{n-1}) \oplus (\lambda_1 \wedge \ell'_0, \ldots, \lambda_1 \wedge \ell'_{n-1}) \tag{5.9}$$

where $\lambda = (\lambda_0,\ \lambda_1) \in \mathbb{F}_2^2 \backslash \{(0,0),(1,1)\}$, and $\bar{k} = (\bar{k}_0,\ \ldots, \bar{k}_{n-1})$, and $\ell' = (\ell'_0, \ldots, \ell'_{n-1})$. Therefore, from the definition of g, we can easily conclude that $\mathbb{K}_i$ contains all the elements of $\mathbb{L}'_{i-1}$, and $\overline{\mathbb{K}}_{i-1}$. Hence, modeling g is equivalent to modeling f_2^i. Now, we construct the linear inequalities whose feasible solutions are precisely the g function trail. To do so, first, we have to build the linear inequalities which are sufficient to describe the propagation $(\mathtt{a},\ \mathtt{b}) \xrightarrow{\wedge} \mathtt{c}$, where $\mathtt{a}$, $\mathtt{b}$, $\mathtt{c} \in \mathbb{F}_2$ are MILP variables corresponding to input and output of the function $\wedge$:

$$\begin{cases} \mathtt{a} - \mathtt{c} & \geqslant & 0 \\ \mathtt{b} - \mathtt{c} & \geqslant & 0 \\ \mathtt{a} + \mathtt{b} - \mathtt{c} & \leqslant & 1 \end{cases} \tag{5.10}$$

Therefore, using Equation 5.9 and Equation 5.10, we can easily conclude that the below inequalities adequately capture the propagation behaviour of g function, i.e., $(\lambda_0, \lambda_1, \bar{k}, \ell') \xrightarrow{g} k$ (where $\mathtt{a}_0$, $\mathtt{a}_1$, $\bar{\mathtt{k}}$, and $\mathtt{l}'$ are MILP variables corresponding to the input of g, and $\mathtt{k}$ is the MILP variable corresponding to the output of g):

$$\begin{cases} \mathtt{a}_0 - \mathtt{p}_j & \geqslant & 0, & \text{for } j = 0, 1, ..., n-1 \\ \bar{\mathtt{k}}_j - \mathtt{p}_j & \geqslant & 0, & \text{for } j = 0, 1, ..., n-1 \\ \mathtt{a}_0 + \bar{\mathtt{k}}_j - \mathtt{p}_j & \leqslant & 1, & \text{for } i = 0, 1, ..., n-1 \\ \mathtt{a}_1 - \mathtt{q}_j & \geqslant & 0, & \text{for } i = 0, 1, ..., n-1 \\ \mathtt{l}'_j - \mathtt{q}_j & \geqslant & 0, & \text{for } i = 0, 1, ..., n-1 \\ \mathtt{a}_1 + \mathtt{l}'_j - \mathtt{q}_j & \leqslant & 1, & \text{for } i = 0, 1, ..., n-1 \\ \mathtt{p}_j + \mathtt{q}_j - \mathtt{k}_j & = & 0, & \text{for } j = 0, 1, ..., n-1 \\ \mathtt{a}_0 + \mathtt{a}_1 & = & 1 \end{cases} \tag{5.11}$$

where $\mathtt{p}_0, \ldots, \mathtt{p}_{n-1}, \mathtt{q}_0, \ldots, \mathtt{q}_{n-1}, \mathtt{l}'_0, \ldots, \mathtt{l}'_{n-1}, \mathtt{k}_0, \ldots, \mathtt{k}_{n-1}, \bar{\mathtt{k}}_0, \ldots, \bar{\mathtt{k}}_{n-1}, \mathtt{a}_0, \mathtt{a}_1$ are binaries and $\mathtt{p} = (\mathtt{p}_0, \mathtt{p}_1, \ldots, \mathtt{p}_{n-1})$, $\mathtt{q} = (\mathtt{q}_0, \mathtt{q}_1, \ldots, \mathtt{q}_{n-1})$ are auxiliary variables. Hence, Equation 5.11 and Equation 5.8 jointly describe the complete MILP model of the `Key-XOR` operation w.r.t BDPT.

5.2.4 MILP Model Construction of r-Round Function

We already describe how to accurately construct a set of linear inequalities depicting one round BDPT propagation for all functions. To build a MILP model for r round BDPT propagation, we must iterate this aforementioned procedure r times. Finally, we conclude upon getting a system of linear inequalities $\mathcal{L}$, which we describe in Algorithm 6.

More precisely, Algorithm 6 constructs a system of linear inequalities which characterizes all r-round BDPT trails i.e.,

$$(k^0 = k, \ell^0 = \ell) \xrightarrow{f_1} (k^1, \ell^1) \xrightarrow{f_2} \ldots \xrightarrow{f_r} (k^r, \ell^r)$$

Finally, we need to construct the MILP model using $\mathcal{L}$, along with appropriate initial and stopping rules and the search algorithm to find an integral distinguisher.

5.3 Automatic Search Algorithm for r-round Integral Distinguisher

This section examines the starting point and stopping criterion for identifying integral distinguishers using BDPT. We obtain a system of linear inequalities $\mathcal{L}$ from Algorithm 6, which depends on input vectors k and ℓ. Next, we convert the stopping criterion into an objective function. By combining $\mathcal{L}$ with this objective function, we construct the MILP model $\mathcal{M}_{\mathbb{K},\mathbb{L}}$. Finally, we propose an algorithm to search for integral distinguishers based on BDPT, starting with the initial BDPT $D^{1^n}_{\{k\},\{\ell\}}$ for an n-bit block cipher, and we demonstrate the correctness of this algorithm.

5.3.1 Initial BDPT

In [21], Todo and Morii initialized the initial BDPT as ($\mathbb{K} = \{\mathbf{1}\}$, $\mathbb{L} = \{7fffffff\}$) when investigating BDPT for SIMON32. They set the active bits of the vector ℓ to 1 and the constant bits to 0. Following their approach, we denote the input BDPT as $k^0 = (k_0^0, k_1^0, ..., k_{n-1}^0)$ and $\ell^0 = (\ell_0^0, \ell_1^0, \ldots, \ell_{n-1}^0)$, with $k^0 = \mathbf{1}$, where n is the block size. Additionally, in our automatic search algorithm, we denote the MILP variables $\mathtt{k}^0$, and $\mathtt{l}^0$ corresponding to the initial BDPT. The constraints on $\mathtt{k}_i^0$ and $\mathtt{l}_i^0$ are

$$\mathtt{k}_i^0 = 1 \quad \text{for } i = 0, 1, ..., n-1$$
$$\mathtt{l}_i^0 = \begin{cases} 1, & \text{if } \ell_i^0 = 1 \\ 0, & \text{otherwise} \end{cases}$$

5.3.2 Stopping Rule

Our automated search model examines the parity of a single output bit, denoted as the q-th bit, for simplicity and clarity. After r rounds, the output set has BDPT $D^{1^n}_{\mathbb{K}_r, \mathbb{L}_r}$. Thus, Proposition 5.4 tells us that the set of first components of the final vectors in all r-round BDPT trails starting from the vector (k, ℓ) is equivalent to $\mathbb{K}_r$. Hence, to check whether there exists any unit vector in the $\mathbb{K}_r$, the objective function can be set as follows:

$$\mathtt{Obj} : \mathtt{Minimize}(\mathtt{k}_0^r + \mathtt{k}_1^r + \ldots + \mathtt{k}_{n-1}^r) \tag{5.12}$$

Similarly, according to Proposition 5.4, the set of the second components of the last vectors of all r-round BDPT trails which start from the vector (k, ℓ) is equal to $\mathbb{L}_r$. Therefore, we can define the objective function as follows:

$$\mathtt{Obj} : \mathtt{Minimize}(\mathtt{l}_0^r + \mathtt{l}_1^r + \ldots + \mathtt{l}_{n-1}^r) \tag{5.13}$$

Now, at first, we construct the MILP model $\mathcal{M}_{\mathbb{K},\mathbb{L}}$ using the system of linear inequalities $\mathcal{L}$ we get from Algorithm 6 and the objective function defined in Equation 5.12. Moreover, we construct another MILP model $\mathcal{M}_{\mathbb{L}}$ as follows:

$$\mathcal{M}_{\mathbb{L}} = \mathtt{ConstructModel}(\mathcal{L}^*, \mathtt{Minimize}(\mathtt{l}_0^r + \ldots + \mathtt{l}_{n-1}^r))$$

where $\mathcal{L}^*$ consists of linear inequalities, defining feasible solutions that precisely match division trails from the set $\mathbb{L}_0$ to $\mathbb{L}_r$.

Stopping Rule for the MILP Model $\mathcal{M}_{\mathbb{K},\mathbb{L}}$.

To check whether $\mathbb{K}_r$ contains the unit vector e_q is equivalent to check whether the MILP model $\mathcal{M}_{\mathbb{K},\mathbb{L}}$ has feasible solution satisfying the constraint $\mathtt{k}^r = e_q$. Therefore, we can set the stopping rule as follows:

$$\mathtt{k}_j^r = \begin{cases} 1 & if\ j = q \\ 0 & otherwise \end{cases} \tag{5.14}$$

If $\mathcal{M}_{\mathbb{K},\mathbb{L}}$ has such feasible solutions, then the q-th output bit is unknown.

Algorithm 6 Computing A Set of Constraints Characterizing BDPT Propagation

Input: The initial input BDPT of an n-bit iterated cipher $D^{1^n}_{\mathbb{K}_0=\{k\},\mathbb{L}_0=\{\ell\}}$
$\mathcal{L}_k(\mathbb{K}_{i-1}, \overline{\mathbb{K}}_{i-1})$: a constraint set consists of linear inequalities, defining feasible solutions that precisely match division trails from the set $\mathbb{K}_{i-1}$ to set $\overline{\mathbb{K}}_{i-1}$, $\forall\, i \in [r]$.
$\mathcal{L}_\ell(\mathbb{L}_{i-1}, \overline{\mathbb{L}}_{i-1})$: a constraint set consists of linear inequalities, defining feasible solutions that precisely match division trails from the set $\mathbb{L}_{i-1}$ to set $\overline{\mathbb{L}}_{i-1}$, $\forall\, i \in [r]$.
$\mathtt{New}_k(\overline{\mathbb{L}}_{i-1}, \mathbb{L}'_{i-1})$: a constraint set consists of linear inequalities, defining feasible solutions that precisely match f_1^i function trails, $\forall\, i \in [r]$.
$\mathtt{Union}_k(\mathbb{L}'_{i-1}, \overline{\mathbb{K}}_{i-1}, \mathbb{K}_i)$: a constraint consists of linear inequalities, defining feasible solutions that precisely match f_2^i function trails, $\forall\, i \in [r]$.
Output: A constraint set of linear inequalities $\mathcal{L}$ describing r-round BDPT propagation.
Begin
$\mathcal{L} = \varnothing$, $\mathcal{C}^i = \bar{\mathcal{C}}^i = \varnothing$ where $i = 1, 2, \ldots, r$
Allocate n-bit MILP variables $\mathtt{k}^i$, $\mathtt{l}^i$ to denote vectors in the set $\mathbb{K}_i$, $\mathbb{L}_i$ respectively where $i = 0, 1, \ldots, r$
Allocate n-bit MILP variables $\bar{\mathtt{l}}^i$, $\mathtt{p}^i$, and $\bar{\mathtt{k}}^i$ to denote vectors in the set $\overline{\mathbb{L}}_i$, $\mathbb{L}'_i$, and $\overline{\mathbb{K}}_i$ respectively where $i = 0, 1, \ldots, r-1$
$\mathcal{L} \leftarrow (\mathtt{k}^0 = k)$, $\mathcal{L} \leftarrow (\mathtt{l}^0 = \ell)$
for $(i = 1\,;\, i \leqslant r\,;\, i{+}{+})$ **do**
    $\mathcal{C}^i \leftarrow \mathcal{L}_\ell(\mathbb{L}_{i-1}, \overline{\mathbb{L}}_{i-1}) \cup \mathcal{L}_k(\mathbb{K}_{i-1}, \overline{\mathbb{K}}_{i-1})$
    $\bar{\mathcal{C}}^i \leftarrow \mathtt{New}_k(\overline{\mathbb{L}}_{i-1}, \mathbb{L}'_{i-1})$
    $\bar{\mathcal{C}}^i \leftarrow \mathtt{Union}_k(\mathbb{L}'_{i-1}, \overline{\mathbb{K}}_{i-1}, \mathbb{K}_i)$
    $\mathcal{L} \leftarrow (\bar{\mathtt{l}}^{i-1} = \mathtt{l}^i)$
    $\mathcal{L} \leftarrow (\mathcal{C}^i \cup \bar{\mathcal{C}}^i)$
return $\mathcal{L}$

Stopping Rule for the MILP Model $\mathcal{M}_{\mathbb{L}}$.

If $\mathbb{K}_r$ does not include e_q, then verifying if $\mathbb{L}_r$ includes e_q is the same as determining whether the MILP model $\mathcal{M}_{\mathbb{L}}$ has a feasible solution that meets the condition $\mathit{l}^r = e_q$. Therefore, we can set the stopping rule as follows:

$$\mathit{l}^r_j = \begin{cases} 1 & if\ j = q \\ 0 & otherwise \end{cases} \tag{5.15}$$

If both $\mathbb{K}_r$ and $\mathbb{L}_r$ do not contain e_q, then q-th output bit is balanced. Otherwise, we need to count the number of feasible solutions satisfying the constraint $\mathit{l}^r = e_q$ of the model $\mathcal{M}_{\mathbb{L}}$. Therefore, the parity of q-th output bit is 0 or 1 if the number of solutions is even or odd respectively, as $\mathbb{K}_r$ does not contain e_q.

5.3.3 Automatic Search Algorithm to Find Integral Distinguishers based on BDPT

We develop an automated search algorithm to find integral distinguishers using BDPT. This approach focuses on identifying whether the q-th output bit's parity can be determined starting with an initial BDPT $D^{1^n}_{\mathbb{K}_0=\{k\},\mathbb{L}_0=\{\ell\}}$ for an n-bit block cipher. Initially, we define various round-specific and auxiliary variables. Following this, we create two MILP models: $\mathcal{M}_{\mathbb{K},\mathbb{L}}$, which captures all r-round BDPT trails, and $\mathcal{M}_{\mathbb{L}}$, which models all r-round division trails for $\mathbb{L}$. By implementing specific starting and termination rules, we can effectively determine the parity of the q-th output bit using BDPT. The complete framework is outlined in Algorithm 7.

5.3.4 Correctness of Search Algorithm

Consider the initial input BDPT of a n-bit iterated cipher as $D^{1^n}_{\mathbb{K}_0=\{k\},\mathbb{L}_0=\{\ell\}}$. The output BDPT is denoted as $D^{1^n}_{\mathbb{K}_r,\mathbb{L}_r}$ after r-round propagation. It is evident that the parity of the q-th bit is *unknown* if e_q is a member of $\mathbb{K}_r$, and 0 if e_q is not a member of $\mathbb{K}_r$ or $\mathbb{L}_r$.
Therefore, to prove the correctness of Algorithm 7, we have to prove that if the q-th unit vector does not belong to $\mathbb{K}_r$ and belongs to $\mathbb{L}_r$, then the parity of q-th output bit is 0 or 1, provided the number of division trails from ℓ to e_q is even or odd, respectively. We first prove the following Lemma:

Lemma 5.1 *Let $\mathbb{X} \subseteq \mathbb{F}_2^n$ has BDPT $D^{1^n}_{\mathbb{K}_0=\{k\},\mathbb{L}_0=\{\ell\}}$ and after r-round propagation, the output set $\mathbb{Y}_r$ has BDPT $D^{1^n}_{\mathbb{K}_r,\mathbb{L}_r}$. For any $\ell' \in \mathbb{L}_r$, if the number of division trails in $\mathbb{L}$ from ℓ to ℓ' is even, then there exists at least one j in $[r]$, such that $\mathbb{L}_j$ contains at least one element u which is produced an even number of times from the elements in $\mathbb{L}_{j-1}$.*

Proof. Let $\mathbb{X} \subseteq \mathbb{F}_2^n$ has division property $D^{1^n}_{\mathbb{K}_0,\mathbb{L}_0=\{\ell\}}$ and after r round propagation, the output set has division property $D^{1^n}_{\mathbb{K}_r,\mathbb{L}_r}$. We know that, every element in $\mathbb{L}_j$ is produced from some elements in $\mathbb{L}_{j-1}$ under the BDPT propagation rules, for all $j \in [r]$.

Assume that the set $\mathbb{L}_j$ contains n_j elements for each $j = 0, 1, 2, \ldots, r$. The number of elements in $\mathbb{L}_{j-1}$ from which any element $v \in \mathbb{L}_j$ is generated by the BDPT propagation rule

Algorithm 7 Deciding Parity of q-th Output Bit

Input: The r-round cipher E_r
The initial input BDPT of an n-bit iterated cipher $D^{1^n}_{\mathbb{K}_0=\{k\},\mathbb{L}_0=\{\ell\}}$
The number q
$\mathcal{L}_\ell(\mathbb{L}_{i-1}, \mathbb{L}_i)$: a constraint set consists of linear inequalities, defining feasible solutions that precisely match division trails from the set $\mathbb{L}_{i-1}$ to set $\mathbb{L}_i$, $\forall\, i \in [r]$.
Output: The data indicating if the q-th output bit is balanced.
Begin
Allocate all the MILP variables denoting the input and output BDPT
$\mathtt{Obj}_1 = \mathtt{Minimize}(\mathtt{k}^r_0 + \mathtt{k}^r_1 + \ldots, \mathtt{k}^r_{n-1})$
$\mathtt{Obj}_2 = \mathtt{Minimize}(\mathtt{l}^{\,r}_0 + \mathtt{l}^{\,r}_1 + \ldots, \mathtt{l}^{\,r}_{n-1})$
Call Algorithm 6 and get a constraint set $\mathcal{L}$ whose feasible solutions are r-round BDPT trail
$\mathcal{M}_{\mathbb{K},\mathbb{L}} = \mathtt{ConstructModel}(\mathcal{L}, \mathtt{Obj}_1)$
$\mathcal{M}_{\mathbb{K},\mathbb{L}} \cdot \mathtt{AddConstraint}(\mathtt{k}^r = e_q)$
if the MILP model $\mathcal{M}_{\mathbb{K},\mathbb{L}}$ has solutions **then**
    return *unknown*
else
    $\mathcal{M}_{\mathbb{L}} = \mathtt{ConstructModel}(\bigcup_{i=1}^{r} \mathcal{L}_\ell(\mathbb{L}_{i-1}, \mathbb{L}_i), \mathtt{Obj}_2)$
    $\mathcal{M}_{\mathbb{L}} \cdot \mathtt{AddConstraint}(\mathtt{l}^{\,0} = \ell)$
    if The MILP model $\mathcal{M}_{\mathbb{L}}$ has no feasible solution satisfying $\mathtt{l}^{\,r} = e_q$ **then**
        return 0
    else
        $\mathcal{M}_{\mathbb{L}} \cdot \mathtt{AddConstraint}(\mathtt{l}^{\,r} = e_q)$
        Count the number of solutions in $\mathcal{M}_{\mathbb{L}}$
        if Count is even **then**
            return 0
        else
            return 1

is referred to as the indegree of v, for all $j \in [r]$. We already assume that, for $j \in [r]$, $\mathbb{L}_j$ contains n_j elements and we denote those elements as $v_1^j, v_2^j, ..., v_{n_j}^j$. For $j = 0$, $\mathbb{L}_0$ contains one element ℓ, i.e., $n_0 = 1$, and for $j = r$, $n_j = 1$. In addition, we assume that, v_i^j has indegree as x_i^j, for all $i \in [n_j]$ and $j \in [r]$. Here, we denote $\mathcal{P}(\ell, v)$ is the number of division trails from the vector ℓ to v.

Now, we prove the above statement by contradiction. Therefore, we assume that, for all $j \in [r]$, all the elements of the set $\mathbb{L}_j$ have indegree as an odd number, i.e., all the elements of the set $\mathbb{L}_j$ are produced an odd number of times from the elements in $\mathbb{L}_{j-1}$. Now we want to prove the statement that the number of division trails in $\mathbb{L}$ from ℓ to v_i^j, for all $i \in [n_j]$, and $j \in [r]$ are odd. We prove this by induction.

Induction Base According to the BDPT propagation in $\mathbb{L}$, as all the elements in $\mathbb{L}_1$ are produced from $\ell \in \mathbb{L}_0$, then

$$x_1^1 = x_2^1 = ... = x_{n_1}^1 = 1.$$

Hence, $\mathcal{P}(\ell,\, v_i^1) = 1$, for all $i \in [n_1]$. Hence, the above statement is true for $j = 1$ and $i \in [n_1]$.

Now, we prove this statement for $j = 2$. According to the BDPT propagation in $\mathbb{L}$, all the elements in $\mathbb{L}_2$ are produced from the elements in $\mathbb{L}_1$. Therefore, for all $i \in [n_2]$, the number of division trails from ℓ to v_i^2 is as follows:

$$\mathcal{P}(\ell,\, v_i^2) = \sum_{i_1 \in I_i^2} \mathcal{P}(\ell,\, v_{i_1}^1),\ \forall i \in [n_2]$$

where $I_i^j \subseteq [n_{j-1}]$ and this set contains the indices k of the elements $v_k^{j-1} \in \mathbb{L}_{j-1}$ from which $v_i^j \in \mathbb{L}_j$ is produced. In addition, the cardinality of I_i^j is x_i^j for $i \in [n_j]$ and $j \in [r]$. Now here, for $j = 2$ and for all $i \in [n_2]$, the cardinality of I_i^2 is odd as the value of x_i^2 is odd as per our assumption.

Hence, $\mathcal{P}(\ell, v_{i_1}^1)$ is odd for all $i_1 \in I_i^2 \subseteq [n_1]$, since it is known that $\mathcal{P}(\ell, v_{i_1}^1) = 1$ for every $i_1 \in [n_1]$. Consequently, $\mathcal{P}(\ell, v_i^2)$ is odd for all $i \in [n_2]$, as the cardinality of I_i^2 is odd. Therefore, the above statement holds true for $j = 2$ and $i \in [n_2]$.

Induction Hypothesis Suppose, the above statement is true for all $j \in [m]$, where $m \leqslant r-1$ and $i \in [n_j]$ i.e.,

$$\mathcal{P}(\ell,\, v_i^j) = \text{odd},\ \ \forall\, i \in [n_j] \text{ and } \forall\, j \in [m] \tag{5.16}$$

Inductive Step Now, we want to prove the above statement for $j = m + 1$ and $i \in [n_{m+1}]$. Therefore, for all $i \in [n_{m+1}]$, the number of division trails from ℓ to v_i^{m+1} is

$$\mathcal{P}(\ell,\, v_i^{m+1}) = \sum_{i_2 \in I_i^{m+1}} \mathcal{P}(\ell,\, v_{i_2}^m)\ \ \forall\, i \in [n_{m+1}]$$

where $I_i^{m+1} \subseteq [n_m]$ and this set contains the indices k of the elements $v_k^m \in \mathbb{L}_m$, from which $v_i^{m+1} \in \mathbb{L}_{m+1}$ is produced. In addition, the cardinality of I_i^{m+1} is x_i^{m+1}, for all $i \in [n_{m+1}]$. Now, for $j = m + 1$ and for all $i = [n_{m+1}]$, the cardinality of I_i^{m+1} is odd as the value of x_i^{m+1} is odd as per our assumption.

Moreover, $\mathcal{P}(\ell,\, v_{i_2}^m)$ is odd for all $i_2 \in I_i^{m+1} \subseteq [n_m]$, since it is known that $\mathcal{P}(\ell,\, v_{i_2}^m)$ is odd for every $i_2 \in [n_m]$ is odd as per induction hypothesis. Therefore $\mathcal{P}(\ell,\, v_i^{m+1})$ is odd, as the

cardinality of I_i^{m+1} is odd, for all $i \in [n_m]$. Hence, the above statement is true for $j = m+1$ and $i \in [n_{m+1}]$.

By induction, we establish that if all elements of the set $\mathbb{L}_j$ have an odd indegree, meaning each element of $\mathbb{L}_j$ is generated an odd number of times from the elements in $\mathbb{L}_{j-1}$, then the number of division trails in $\mathbb{L}$ from ℓ to v_i^j is odd for all $i \in [n_j]$ and $j \in [r]$. Therefore, the number of division trails from ℓ to v_1^r is odd, which contradicts the given information that the number of division trails in $\mathbb{L}$ from ℓ to ℓ' is even. Hence, our assumption is incorrect.

Thus, there must exist at least one $j \in [r]$ such that $\mathbb{L}_j$ contains at least one element v that is generated an even number of times from the elements in $\mathbb{L}_{j-1}$, thereby completing our proof.

□

Therefore, using Lemma 5.1 we prove the final result as follows:

Proposition 5.5 *Let $\mathbb{X} \subseteq \mathbb{F}_2^n$ has division property $D^{1^n}_{\mathbb{K}_0=\{k\},\mathbb{L}_0=\{\ell\}}$ and after r-round propagation, the output set $\mathbb{Y}$ has division property $D^{1^n}_{\mathbb{K}_r,\mathbb{L}_r}$. If e_q doesn't belong to the set $\mathbb{K}_r$, where $q \in [n]$, then we have:*

1. *If the number of division trails from ℓ to e_q is even in $\mathbb{L}$, then $\bigoplus_{y\in\mathbb{Y}} y_q = 0$.*
2. *If the number of division trails from ℓ to e_q is odd in $\mathbb{L}$, then $\bigoplus_{y\in\mathbb{Y}} y_q = 1$.*

Proof. Let $S \subseteq (\mathbb{F}_2^n)^{r+1}$ be the set which contains all the division trail in $\mathbb{L}$ from ℓ to e_q and $|S|$ is even. Now, by using Lemma 5.1, we can easily conclude that there exists at least one $j \in \{2, 3, ..., r\}$ such that $\mathbb{L}_j$ contains an element u which is produced an even number of times from the elements in $\mathbb{L}_{j-1}$. Without loss of generality, we choose the smallest such j.

According to the BDPT propagation rule of `XOR` and S-box, we can see that if an element u is produced an even number of times in $\mathbb{L}_j$ from $\mathbb{L}_{j-1}$, then the following holds:

$$\bigoplus_{y\in\mathbb{Y}_j} y^u = 0$$

where $\mathbb{Y}_j$ is the multi set whose BDPT is $D^{1^n}_{\mathbb{K}_j,\mathbb{L}_j}$ and that implies u should not be in $\mathbb{L}_j$. Hence, all the division trails from ℓ to e_q, which contains the vector u, are redundant, and the number of redundant division trails must be even. Therefore, we can remove these redundant division trails from S and call the new set S_1. It is trivial that either $|S_1|$ is even or $|S_1| = 0$.
Case-I. If $|S_1| = 0$, then it implies that all the division trails from ℓ to e_q contains the element u. Therefore, as u shouldn't be in $\mathbb{L}_j$, so e_q also shouldn't be in $\mathbb{L}_r$ and it is given that e_q doesn't belong to $\mathbb{K}_r$ which means

$$\bigoplus_{y\in\mathbb{Y}} y^{e_q} = \bigoplus_{y\in\mathbb{Y}} y_q = 0$$

Case-II. If $|S_1|$ is even, then in a similar way, we can find an even number of redundant division trails from ℓ to e_q in $\mathbb{L}$ and construct S_2 from S_1 where $|S_2|$ is either even or 0 and so on.

As $|S|$ is finite, then after finitely many p steps, we must get some S_p such that, $|S_p| = 0$. Hence, e_q should not be in $\mathbb{L}_r$ and it is given that e_q does not belong to $\mathbb{K}_r$ which means

$$\bigoplus_{y\in\mathbb{Y}} y^{e_q} = \bigoplus_{y\in\mathbb{Y}} y_q = 0$$

which completes the first part of the proof.
Now, it is given that the number of division trails in $\mathbb{L}$ from ℓ to e_q is odd. Similarly, we can construct a set S' containing all such division trails. Therefore, there may or may not exist $j \in \{2, 3, ..., r\}$ s.t $\mathbb{L}_j$ contains an element u which is produced an even number of times from the elements in $\mathbb{L}_{j-1}$.
Case-A If there doesn't exist any such j, then by BDPT propagation rules, we can easily conclude that no division trail from ℓ to e_q is redundant. Therefore, it implies that e_q belongs to $\mathbb{L}_r$ which means $\bigoplus_{y\in\mathbb{Y}} y_q = 1$.
Case-B If there exist some j s.t $\mathbb{L}_j$ contains an element u which is produced an even number of times from the elements in $\mathbb{L}_{j-1}$. Similarly, by the previous argument, we can easily conclude that all the division trails from ℓ to e_q, which contains u, are actually redundant. Therefore, we can remove these redundant division trails from S' and call the new set S'_1. It is evident that $|S'_1|$ is odd.

Now, continuing this way, after finitely many steps, we arrive at a situation where the number of remaining division trails from ℓ to e_q is odd, and no redundant division trails are left, which implies e_q belongs to $\mathbb{L}_r$. Therefore, $\bigoplus_{y\in\mathbb{Y}} y^{e_q} = \bigoplus_{y\in\mathbb{Y}} y_q = 1$ which completes the second part of the proof. □

5.4 Applications to Block Ciphers

In this section, we apply our automated search algorithm for BDPT to several block ciphers: SIMON, SIMON(102), MANTIS, PRINCE, KLEIN, and PRIDE. All experiments[2] were conducted on an Intel Core i5-8250U CPU running at 1.60GHz with 8GB RAM, using 64-bit Ubuntu 18.04.5 LTS. The optimizer used for solving MILP models was Gurobi 9.1.2 [145].

For the integral distinguishers, "?" denotes bits with unknown balanced information, "0" represents bits with a sum of zero, and "1" indicates bits with a sum of one. Below are the detailed integral distinguishers for PRINCE, MANTIS, KLEIN, and PRIDE.

5.4.1 Applications to PRINCE and MANTIS

In this section, we present the application of our BDPT model to the cipher PRINCE and MANTIS. It is important to note that, MANTIS uses a binary matrix to perform the mix-column operations, but PRINCE does not have binary matrix as their mixcolumn operation. Instead of that, the matrix used in mixcolumn operation in PRINCE has the property that the rows of the matrix can be grouped into cosets in such a way that the rows in different cosets do not share nonzero entries in the same column. Hence, we apply our method to model the binary linear layer in BDPT and construct the MILP model efficiently. Then, choosing appropriate initial BDPT, we find improved integral distinguisher as follows:

[2]The codes are publicly available: https://github.com/Debasmita-isi/three-subset-division-property

Table 5.5: Integral Distinguishers of PRINCE

Cipher	Distinguisher	Ref
(2+2)-PRINCE64	In: (1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1110) Out: (0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000)	BDPT

Integral Attack on PRINCE.

Block ciphers utilizing the reflection design strategy, pioneered by PRINCE [25], are widely favored in low-latency design applications. PRINCE is a 64-bit block cipher which uses 128-bit key. A brief description of PRINCE has been illustrated in Subsection 2.4.8.

Let the total number of rounds in PRINCE be represented as $a + b$, where a denotes rounds preceding the middle layer, and b denotes rounds following it. There are several attacks (Integral attack, higher order differential attack, boomerang attack) on PRINCE [146]–[148]. Now, in [142], the authors applied CBDP on PRINCE and found $2 + 1$ and $1 + 2$ round integral distinguishers which are best integral distinguisher till date.

For PRINCE, we identify an integral distinguisher spanning $2 + 2$ rounds, surpassing the previous best findings by one round [142].

Integral Attack on MANTIS.

MANTIS is a tweakable block cipher introduced by Beierle *et al.* in [26]. Structurally, it resembles PRINCE (for a concise specification, see Subsection 2.4.9). This cipher operates on a 64-bit message block, employs a 64-bit tweak, and uses a $(64 + 64)$-bit key. In the light of cryptanalysis, there are several attacks [149]–[151] on MANTIS. For MANTIS, we find a $3 + 3$ round integral distinguisher based on BDPT which is one more round than the previous best results [142].

5.4.2 Applications to KLEIN and PRIDE

To complete our BDPT analysis on ciphers with complex linear layers, we apply our automatic search algorithm for BDPT to block ciphers KLEIN and PRIDE which have non-binary linear layers. To handle non-binary linear layers, we trivially decompose the linear layers as `COPY` and `XOR` operations and construct the MILP model accordingly. Then, choosing an appropriate initial BDPT, we find the integral distinguisher as follows:

Table 5.6: Integral Distinguishers of MANTIS

Cipher	Distinguisher	Ref
(3+3)-MANTIS64	In: (1011, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111) Out: (0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000)	BDPT

Table 5.7: Integral Distinguishers of KLEIN

Cipher	Distinguisher	Ref
6-KLEIN64	In: (1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 0011, 1111, 1111, 1111, 1111) Out: (0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000, 0000)	CBDP, BDPT

Integral Attack on KLEIN.

The KLEIN family of block ciphers, as described in Gong *et al.* [27], features a consistent 64-bit block size and supports variable key lengths of 64, 80, or 96 bits. Its structure adheres to a standard Substitution Permutation Network (SPN) design (see Figure 2.18). Please consult Subsection 2.4.10 for further specifics.

In the light of cryptanalysis, there are several attacks [144], [152]–[154] on the block cipher KLEIN, mostly on KLEIN-64 (key length 64 bits). In [144], the authors have presented a 5-round integral distinguisher, marking it as the most effective integral distinguisher currently identified. First, we apply MILP-based CBDP on KLEIN and find a 6 round integral distinguisher that is one more round than the previous best results [144]. Therefore, we apply the MILP based BDPT on KLEIN and the integral distinguishers we identify align with those derived from CBDP.

Table 5.8: Integral Distinguishers of PRIDE

Cipher	Distinguisher	Ref
9-PRIDE64	In: (0111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111, 1111) Out: (00??, 00??, 00??, 00??, 00??, 00??, 00??, 00??, 00??, 00??, 00??, 00??, 00??, 00??, 00??, 00??)	CBDP, BDPT

Integral Attack on PRIDE.

PRIDE, depicted in Figure 2.19, is a lightweight block cipher developed by Albrecht *et al.* [28], unveiled at CRYPTO 2014. It operates on an SPN structure with a 64-bit block size and a 128-bit key. Please refer to Subsection 2.4.11 for more detailed specifications. Regarding cryptanalysis, PRIDE has been subject to various attacks [155]–[158]. Initially, we employ MILP-based techniques to analyze CBDP on PRIDE, uncovering a 9-round integral distinguisher, surpassing the previously established best results by one round [143]. Therefore, we apply the MILP-based BDPT on PRIDE and the integral distinguishers we identify align with those derived from CBDP.

5.4.3 Applications to SIMON, SIMON(102)

We apply our method to all versions of SIMON [14] and SIMON(102) [141] block ciphers. The distinguishers we discovered align with previous longest distinguishers [24], but we achieved these results more efficiently. The automatic search algorithm we have created is used on the block cipher SIMON, SIMON(102). We then compare our runtime to that of the algorithm talked about in [24] and [29] in Table 5.9. The round keys in the SIMON and SIMON(102) block ciphers are incorporated by XORing them into the state after each round function. Consequently, we can include an additional round in the distinguishers using this approach in [159], but our method can not directly find these extended integral distinguishers.

5.5 Conclusion

In this chapter, we provide an idea to model BDPT propagation of ciphers with binary (complex) linear layers and construct an efficient automatic search algorithm that accurately characterizes BDPT propagation. Based on these, more accurate BDPT can be obtained for ciphers with complex linear layers such as PRINCE, MANTIS, KLEIN, PRIDE.

Table 5.9: Summarization of Integral Distinguishers of SIMON, and SIMON(102)

Cipher	Data	#R	Constant Bits	Time	Ref.
SIMON32	2^{31}	15	3	1.6min	[24]
	2^{31}	15	3	**1.3**min	**This work**
SIMON48	2^{47}	16	24	8.4min	[24]
	2^{47}	15	24	**6**min	**This work**
SIMON64	2^{63}	18	27	1hr8min	[24]
	2^{63}	18	27	**25**min	**This work**
SIMON96	2^{95}	22	5	5hr55min	[24]
	2^{95}	22	5	**1hr30min**	**This work**
SIMON128	2^{127}	26	3	21hr7min	[24]
	2^{127}	26	3	**3hr50min**	**This work**
SIMON(102)32	2^{31}	20	3	-	[24]
	2^{31}	20	3	22min	[29]
	2^{31}	20	3	**2min**	**This work**
SIMON(102)48	2^{47}	28	3	-	[24]
	2^{47}	28	3	1hr10min	[29]
	2^{47}	28	3	**15min**	**This work**
SIMON(102)64	2^{63}	36	3	-	[24]
	2^{63}	36	3	3hr27min	[29]
	2^{63}	36	3	**45min**	**This work**

Chapter 6

Superpoly Recovery of Grain-128AEAD Using Division Property

Dinur and Shamir introduced the cube attack [31], presenting it as a formidable technique for cryptanalysis against symmetric cryptographic systems. The objective of the cube attack is to extract secret variables from the simplified polynomial representation of the ANF of a Boolean function known as the superpoly.

In various studies [31], [160]–[162], ciphers are treated as black boxes, and superpolies are recovered through experimental methods. The study by Todo *et al.* [163] revolutionized the handling of polynomials, adopting a non-blackbox approach and integrating CBDP into cube attacks on stream ciphers, representing a groundbreaking innovation in cryptographic techniques.

Following the previous work, Wang *et al.* [164] further advanced the approach by integrating flag and term enumeration techniques. Facilitating the theoretical retrieval of superpolies from large cubes using CBDP-based cube attacks, this advancement represents a significant step forward. However, according to CBDP theory, there is no assurance that the superpoly of a cube will not be constant. Consequently, the attack designed for key recovery may function primarily to distinguish cryptographic behaviour. To address this issue, Wang *et al.* [23] introduced the cube attack utilizing the BDPT and demonstrated that BDPT, when lacking an unknown subset, can accurately recover the superpoly in a cube attack. Subsequently, Hao *et al.* [33] presented a novel modeling technique for BDPT, excluding an unknown subset. Their method improves efficiency and significantly enhances key-recovery attacks on various cryptographic ciphers. Furthermore, researchers in [34], [35] employed a nested framework incorporating the monomial prediction technique to recover superpolies. Additionally, in [162], a purely algebraic method was devised to retrieve the superpoly accurately. Yet, as the cipher's number of rounds increases, locating such beneficial cubes becomes increasingly challenging.

An essential security requirement for a cryptographic algorithm with a key is its capability to demonstrate unpredictable behaviour when any key is chosen randomly from the entire range of possible keys. A key is deemed *weak* when it causes a cipher to exhibit undesirable behaviour, such as significantly reducing its algebraic degree. Several attacks in the context of weak keys have been proposed for both block ciphers [165], [166], and stream ciphers [167], [168]. Nevertheless, identifying a weak-key set poses a challenging computational problem.

For instance, the invariant subspace attack [169], [170] is a widely recognized weak-key attack discussed in the literature. Cube attacks examining key conditions that could potentially lead to weak-key vulnerabilities have been proposed in [171], [172].

Table 6.1: Previous Works of Superpoly Recovery for Grain-128AEAD using Division Property

Round	#Cubes	Cube Size	Time	Reference
190	–	95	–	[33]
191	2	$95-96$	–	[34]
192	1	94	45 days	[35]

Related Works. In response to the growing demand for cryptographic solutions tailored to resource-constrained environments, the National Institute of Standards and Technology (NIST) initiated the Lightweight Cryptography (LWC) standardization process in August 2018. The primary goal of this initiative was to identify and standardize lightweight cryptographic algorithms that offer a suitable balance of security, performance, and efficiency for deployment in constrained devices such as IoT sensors, RFID tags, and embedded systems.

As part of this effort, NIST issued a call for cryptographic algorithms, inviting submissions from researchers worldwide. The competition received 57 candidate algorithms, which were rigorously analyzed through multiple rounds of evaluation. The assessment focused on various aspects, including resistance to cryptanalytic attacks, hardware and software performance, energy efficiency, and ease of implementation.

After three rounds of extensive evaluation, in February 2023, NIST announced ASCON as the selected algorithm for standardization. ASCON, originally designed in 2014 as a lightweight authenticated encryption scheme, demonstrated strong security properties, high efficiency across different platforms, and resilience against emerging cryptanalytic techniques.

Among the ten finalist algorithms considered in the third round was Grain-128AEAD, a stream cipher-based authenticated encryption scheme designed for constrained environments. Grain-128AEAD builds upon the well-established Grain family of stream ciphers and was analyzed extensively for its security properties, resistance to cryptanalytic attacks, and implementation efficiency. While it was not ultimately selected as the NIST standard, its inclusion in the final evaluation phase highlights its robustness and relevance in lightweight cryptographic applications.

Since Grain-128AEAD is developed by altering the authentication module of Grain-128a, the cryptanalysis techniques used for the encryption module of Grain-128a are applicable to the encryption module of Grain-128AEAD. There are several cube attacks on Grain-128a and Grain-128AEAD. For a detailed overview, the readers are requested to follow [173]–[181]. Moreover, using the concept of division property-based cube attacks, exact superpolies for 190, 191, 192-round Grain-128AEAD have been recovered efficiently using which key-recovery attacks are also mounted [33]–[35] (The results we have, are listed in Table 6.1). But, for these cube attacks, the cube dimensions are on the higher side. Now, the following question arises in our mind:

Can we reduce the cube dimension and recover the exact superpoly of the cube for more rounds of Grain-128AEAD?

Contributions of this chapter. To address this question, we begin by studying the most popular cipher Grain-128AEAD in the light of division property-based cube attacks. In this chapter, we present a novel approach to division property-based cube attacks on Grain-128AEAD, with the primary goal of reducing the cube dimension and efficiently recovering exact superpolies for higher rounds of the cipher. Unlike previous works, which struggled with increasing rounds due to computational constraints to recover exact superpoly, our approach introduces several improvements that allow us to push the attack further. Below, we outline our key contributions in detail.

1. **Identification of Lower-Dimensional Cubes for Efficient Superpoly Recovery**: In previous division property-based cube attacks on Grain-128AEAD, the selection of cube variables was a critical challenge. The attack's success largely depends on finding cubes that yield a well-structured and computable superpoly.

   - Prior works often relied on higher-dimensional cubes, typically exceeding a size of 94, which made superpoly recovery computationally expensive and impractical for higher rounds. For instance, in the case of 192 rounds of Grain-128AEAD, a cube of size 94 required approximately 45 days to recover the exact superpoly. Further details on previous work are provided in Table 6.1.
   - In contrast, we identify new cubes with dimensions 91, 92, 93, and 94, which were previously unexplored in the context of division property-based cube attacks on Grain-128AEAD (illustrated in Section 6.1.1).

   By systematically searching for cubes with reduced dimensions, we improve the practicality and efficiency of the attack, extending the number of rounds that can be analyzed.

2. **Development of an Algorithm for Conditional Key Bit Search:** In this work, we introduce an algorithm (Algorithm 8) to systematically search for conditional key bits that depend on cube variables.

   - In many cases, the presence of unknown key bits complicates the algebraic structure of the superpoly, making exact recovery challenging.
   - By carefully setting key bit conditions (Section 6.1.2), we simplify the superpoly, making it possible to recover its exact form.
   - Previous superpoly recovery methods using division property-based cube attacks did not impose any conditions on key bits to simplify the structure of the superpoly associated with the cube variables.
   - Our algorithm systematically determines conditions on key bits, ensuring that superpolies can be efficiently computed for the cubes we identified.

3. **Exact Superpoly Recovery in Weak-Key Setting for Higher Rounds (192–195)** One of the most significant contributions of our work is that we achieve exact superpoly recovery for rounds 192–195 of Grain-128AEAD in the weak-key setting.

- Previous research struggled to recover exact superpolies beyond a certain number of rounds due to the complexity of the algebraic expressions involved. The best-known prior results could not systematically recover superpolies for rounds 192 and beyond.
- By leveraging our newly identified low-dimensional cubes and applying the conditional key bit search algorithm, we successfully extend the attack to higher rounds than what was previously possible.
- Precisely, we achieve the best-known results[1] for Grain-128AEAD by identifying exact superpolies in the weak-key setting for rounds 192 to 195.
- We also unveil a novel zero-sum distinguisher applicable to the 193-round version of Grain-128AEAD, representing a significant advancement in employing division property-based cube attacks for distinguishing this cipher. The detailed results are shown in Table 6.2.

These contributions collectively represent a significant advancement over prior division property-based cube attacks on Grain-128AEAD, enabling deeper cryptanalysis of the cipher for higher rounds and leading to the best-known results in this domain.

Table 6.2: Summary of our Superpoly Recovery Results for Grain-128AEAD in the Weak-Key Setup using Division Property

Round	#Cubes	Cube Size	Time	Reference
192	2	91, 92	2min	This work
193	2	92, 94	7min	This work
194	1	93	1hr	This work
195	1	94	7days	This work

Organization. This chapter is organized as follows: In Section 6.1, we construct good cubes and propose an algorithm to construct appropriate weak-key conditions to perform cube attack on Grain-128AEAD. Therefore, we show some results (superpoly recovery, zero-sum distinguisher) on Grain-128AEAD in Section 6.2. At last, we conclude the chapter in Section 6.3.

6.1 Superpoly Recovery for Grain-128AEAD using Weak Keys

As Grain-128AEAD is a finalist in a recent NIST competition, it will be challenging to recover the superpoly of such cipher. Before recovering the superpoly, one needs to search for a good cube of the cipher. If one works on a weak-key setting, another important task is finding conditional key variables, which leads to the recovery of the superpoly of the cube.

[1]We used the tool used in the paper [33], which can be found in https://github.com/ysktodo/milp-three-subset-wo-unknown

6.1.1 Cube Searching Algorithm for Grain-128AEAD

Finding an appropriate cube is crucial in cube attacks, as the effectiveness of the attack largely depends on exploiting the non-randomness properties of the superpoly at lower rounds. Previous studies have explored different techniques to identify such cubes.

In cube attacks, bias refers to the deviation of the superpoly from behaving like a truly random Boolean function. A random Boolean function produces 0 and 1 with equal probability (i.e., 0.5 for each). However, when a function exhibits bias, the probabilities deviate from this uniform distribution, indicating some underlying structural nonrandomness in the cipher. A highly biased superpoly is a superpoly whose output is significantly skewed towards either 0 or 1. That is, if a superpoly outputs 1 with a probability close to 0 or close to 1 (far from 0.5), it is considered highly biased. Such biases can be exploited in cryptanalysis to construct cube testers, key recovery attacks, or distinguishers, as they indicate deviations from ideal cryptographic behavior.

Maximum Last Zero Technique: The Maximum Last Zero technique builds upon the concept of selecting cubes that generate a zero superpoly for the longest possible round range. This approach extends the idea of Maximum Initial Zero, where cubes are chosen based on the longest consecutive rounds for which their superpolies are zero. In contrast, Maximum Last Zero specifically targets the highest round where a cube still produces a zero superpoly. By identifying such cubes, we can construct cube testers that extend to deeper rounds, making them useful for distinguishing attacks and key recovery. This technique has been successfully applied to analyze stream ciphers like Trivium and Trivia-SC.

Maximum Last α Technique: The Maximum Last α technique generalizes the Maximum Last Zero approach by considering cubes whose superpoly exhibits a strong bias at higher rounds rather than strictly being a zero function. Specifically, for some cubes, the superpoly evaluates to 1 with very low probability (denoted as α). The intuition behind this method is that such cubes can potentially propagate bias into later rounds, making them useful for distinguishing attacks. Identifying cubes with highly biased superpolies provides an alternative way to locate exploitable non-randomness in cryptographic primitives, aiding in both structural analysis and cryptanalysis.

Nowadays, constructing a cube-searching algorithm is crucial for a cube attack. Many such algorithms exist for maximum last zero and maximum last α ($0 \leqslant \alpha \leqslant 1$). The previous method gives a better cube searching for Grain-128a. So, we have used this algorithm to find a better result. In the paper [179], the authors have found a cube of $\{63, 64, 66, 68, 69\}$ of size five to mount a distinguishing attack for 191 round in a single key scenario. They construct a cube of size five from a cube of size one.

Following a similar method, we also find a cube of size one. The best cube of size one is $\{s_{69}\}$ because it attains the maximum last alpha round at 123. By similar process, we get the cube variables s_{68}, s_{67}, s_{66}, s_{65}, s_{64} simultaneously. As those cube variables expose some weakness of the Boolean functions at particular rounds, we again work with those cube variables. The abovementioned variables are crucial in getting a distinguisher for Grain-128a. However, when the goal shifts from merely constructing a distinguisher to recovering the

superpoly via the division property, a different approach becomes necessary. We know that a small dimensional cube will not be useful for superpoly recovery in the division property-based cube attacks. Due to the success of our cube variables in the previous attack on Grain-128a, we decide to work with the complement of the set of cube variables. In summary, while the original cube variables were useful for mounting a distinguisher due to the weaknesses the authors [179] revealed, the complementary set was chosen for superpoly recovery because it provides a larger cube, which is crucial for the success of the division property-based cube attack. As superpoly searching is a lengthy and time-consuming process, we start to find the superpoly using a cube of size $96 - 2 = 94$. As previously, the cube of sizes $96, 95$ were used, so we used a cube of less size to reduce the complexity of superpoly recovery. Then, we decrease the cube size one by one, following less complexity for superpoly recovery. Also, we vary the initialization round to reduce the complexity. Finally, we find the best trade-off between the initialization round and cube size to achieve better complexity.

Algorithm 8 Searching for Conditional Bits Corresponding to Chosen Cubes

Input: Set of strong variables $\mathcal{S}$
Output: Set of Conditional key variables $\mathcal{W}$
Begin
Start with a single element from $\mathcal{S}$ and store it in $\mathcal{C}$
while $|\mathcal{C}| \leqslant |\mathcal{S}|$ **do**
    Choose the cube variables as $IV \backslash \mathcal{C}$
    Store the conditional key variables from SAGE corresponding to variables in $IV \backslash \mathcal{C}$ in $\mathcal{W}$.
    Also, store conditional key variables from structure observation of the cipher in $\mathcal{W}$
    Run division property-based cube attacks using the cube $IV \backslash \mathcal{C}$
    if Superpoly corresponding to $IV \backslash \mathcal{C}$ is recovered **then**
        Take $\mathcal{W}$ as a set of conditional key bits
    else
        Add some additional, conditional key variables in $\mathcal{W}$
        Run division property-based cube attacks
        Repeat Else part until superpoly is recovered
    Take another subset $\mathcal{C}$ of $\mathcal{S}$ and repeat the while part.
return $\mathcal{W}$

6.1.2 Searching Weak-Key Domain for Grain-128AEAD

Putting conditions on key and IV variables is important in upgrading the attacks on any cipher. Conditions on the variables help us find weaknesses in the corresponding Boolean function at a particular round. In the previous paper [179], the authors found the conditions on key bits corresponding to cube variables using SAGE software. Also, some conditions are found using structural observation with theoretical analysis. We have also followed their approaches. However, the conditions retrieved for corresponding cubes do not help us to recover superpoly using division property-based approaches. So again, we try to find

the additional conditions to recover the superpoly. We try to find the subset of key bits contributing to superpoly recovery. As the division property-based attack takes all IV bits as zero, we do not worry about the conditions on IV bits. The selection of key bits is made in the following way:

We have implemented the algorithm on the different rounds of Grain-128AEAD in the following way.

- We collect the strong variables[2] of Grain-128AEAD as $\mathcal{S} = \{s_{42}, \cdots, s_{69}\}$.
- The conditions on key variables for each strong variable are given in Table 6.3.
- Select a element say r from $\mathcal{S}$ and take $IV \backslash \{r\}$ as cube. Therefore, collect all corresponding conditional key bits from Table 6.3 in the set $\mathcal{W}$ corresponding to the chosen cube.
- Also, we collect the conditional key variables getting through the structure observation of Grain-128AEAD in the set $\mathcal{W}$ (Given at the end of this section).
- For example, we take $IV \backslash \{s_{69}, s_{68}\}$ as a cube for 195-round Grain-128AEAD and run the division property-based cube attacks to recover the superpoly.
- As we can not recover the superpoly, we add some additional conditional key variables $b_{42}, b_{43}, b_{44}, b_{45}, b_{72}, b_{73}, b_{76}, b_{77}, b_{121}, b_{122}, b_{123}, b_{124}, b_{126}, b_{127}$ in the set $\mathcal{W}$.
- Again, we run the program. This time, we recover the superpoly for 195-round Grain-128AEAD. Similarly, we find $\mathcal{W}$ for different cubes and recover superpolies.

As running a division property-based cube attack is time-consuming, we optimize the $\mathcal{W}$ set as much as possible.

From the structure observation, the additional conditional key bits for the above cubes are $b_{64}, b_{67}, b_{70} - b_{74}, b_{76} - b_{87}, b_{91}, b_{94}, b_{95}, b_{102}, b_{104}, b_{105}, b_{108}, b_{110}, b_{112} - b_{114}, b_{116}, b_{118}, b_{119}, b_{121}, b_{122}, b_{125}$.

6.1.3 Division Property-Based Cube Attack for Grain-128AEAD

The crucial step in a cube attack is recovering the superpoly, referred to as *superpoly recovery* in this chapter. According to [33], using three-subset division property without unknown subset can effectively analyze the ANF coefficients of the superpoly for a given public Boolean function.

Superpoly Recovery.

The encryption mechanism of Grain-128AEAD is modeled as a Boolean function $f(x, v)$, where x represents secret variables and v represents public variables. We formulate a MILP model $\mathcal{M}$ based on the division property, following Algorithm 5 in [33]. In our formulation, $x = (b_0^0, \ldots, b_{127}^0)$ and $v = (s_0^0, \ldots, s_{127}^0)$. Specifically, elements of v indexed by $\mathcal{I}$ (cube

[2]The set of those variables in IV using which we can construct good cubes for Grain-128AEAD.

Table 6.3: Conditions on key variables for 1-dimensional cubes

Cube	Conditions on key variables	Cube	Conditions on key variables
$\{s_{42}\}$	$b_{46} = b_{50} = b_{95} = 0$	$\{s_{56}\}$	$b_{60} = b_{64} = b_{109} = 0$
$\{s_{43}\}$	$b_{47} = b_{51} = b_{96} = 0$	$\{s_{57}\}$	$b_{61} = b_{65} = b_{110} = 0$
$\{s_{44}\}$	$b_{48} = b_{52} = b_{97} = 0$	$\{s_{58}\}$	$b_{62} = b_{66} = b_{111} = 0$
$\{s_{45}\}$	$b_{49} = b_{53} = b_{98} = 0$	$\{s_{59}\}$	$b_{63} = b_{67} = b_{112} = 0$
$\{s_{46}\}$	$b_{50} = b_{54} = b_{99} = 0$	$\{s_{60}\}$	$b_{64} = b_{68} = b_{113} = 0$
$\{s_{47}\}$	$b_{51} = b_{55} = b_{100} = 0$	$\{s_{61}\}$	$b_{65} = b_{69} = b_{114} = 0$
$\{s_{48}\}$	$b_{52} = b_{56} = b_{101} = 0$	$\{s_{62}\}$	$b_{66} = b_{70} = b_{115} = 0$
$\{s_{49}\}$	$b_{53} = b_{57} = b_{102} = 0$	$\{s_{63}\}$	$b_{67} = b_{71} = b_{80} = b_{116} = 0$
$\{s_{50}\}$	$b_{54} = b_{58} = b_{103} = 0$	$\{s_{64}\}$	$b_{68} = b_{72} = b_{117} = 0$
$\{s_{51}\}$	$b_{55} = b_{59} = b_{104} = 0$	$\{s_{65}\}$	$b_{69} = b_{73} = b_{118} = 0$
$\{s_{52}\}$	$b_{56} = b_{60} = b_{105} = 0$	$\{s_{66}\}$	$b_{70} = b_{74} = b_{119} = 0$
$\{s_{53}\}$	$b_{57} = b_{61} = b_{106} = 0$	$\{s_{67}\}$	$b_{71} = b_{75} = b_{120} = 0$
$\{s_{54}\}$	$b_{58} = b_{62} = b_{107} = 0$	$\{s_{68}\}$	$b_{72} = b_{76} = b_{121} = 0$
$\{s_{55}\}$	$b_{59} = b_{63} = b_{108} = 0$	$\{s_{69}\}$	$b_{73} = b_{77} = b_{122} = 0$

indices) are constrained to 1, while elements indexed by other IV indices are constrained to 0 to reflect the initial division property. Additionally, we incorporate constraints in $\mathcal{M}$ that account for weak-key conditions.

After formulating the MILP model $\mathcal{M}$ with initial constraints representing cube and non-cube indices, along with incorporating weak-key conditions, we proceed to solve $\mathcal{M}$. In this process, we utilize Algorithm 2 from [33] to identify all monomials potentially contributing to the superpoly. Subsequently, we enumerate the feasible solutions and derive the superpoly of Grain-128AEAD corresponding to the cube $C_{\mathcal{I}}$, where $\mathcal{I}$ denotes the cube indices. While this method accurately identifies the superpoly for $C_{\mathcal{I}}$, practical limitations arise when attempting to enumerate all feasible solutions, especially in scenarios with numerous solutions.

Upon successful recovery of the superpoly, an attacker gains insight into the Boolean function used within the cipher. Also, the attacker can use the drawbacks in the superpoly to find loopholes in the output function of the cipher, which leads to a distinguishing attack. Further, one can extend it to a key recovery attack using a sufficient number of superpolies.

6.2 Experimental Results

We utilize cube attacks based on the three-subset division property without an unknown subset on the encryption module of Grain-128AEAD within a weak-key setting. Initially, we identify suitable cubes and weak keys using Algorithm 8. Then, through division property-based cube attack techniques, we precisely recover the superpolies for 192-195 rounds, employing cube sizes of 91, 92, 93, 94, respectively, within a weak-key setting where the size of the corresponding weak-key class is 2^{43}. The specifics of our findings are presented in Table 6.4. These represent the most advanced attacks on Grain-128AEAD in a weak-key

setting to date, best known to us. Additionally, we develop zero-sum distinguishers for 192-193 rounds of Grain-128AEAD in a weak-key setting, which are the longest distinguishers of their kind. The detailed parameters for the *superpoly recovery* of 192-round, 193-round, and 194-round Grain-128AEAD and zero-sum distinguishers are discussed in the following subsections.

Analyzing 192-Round Grain-128AEAD: Superpoly Recovery

The cube indices used to retrieve the superpoly for the 192-round Grain-128AEAD were $\mathcal{I} = \{1, 2, \ldots, 65, 71, \ldots, 96\}$, with $IV_{66} = IV_{67} = IV_{68} = IV_{69} = IV_{70} = 0$. This led to identifying the superpoly corresponding to $C_{\mathcal{I}}$, described as follows:

$$p(x) = x_{40}x_{42} + x_{29},$$

where $x = (x_1, x_2, \ldots, x_{128})$ represents the secret key ($x_i = k_i$).

Analyzing 193-Round Grain-128AEAD: Superpoly Recovery

The cube indices used to extract the superpoly for the 193-round Grain-128AEAD were $\mathcal{I} = \{1, 2, \ldots, 66, 71, \ldots, 96\}$, with $IV_{67} = IV_{68} = IV_{69} = IV_{70} = 0$. This resulted in identifying the superpoly corresponding to $C_{\mathcal{I}}$, which consists of the sum of 38 monomials:

$$\begin{aligned} p(x) =& 1 + x_{43} + x_{42}x_{43} + x_{41} + x_{40}x_{42}x_{43} + x_{39}x_{41} + x_{39}x_{40} \\ &+ x_{38} + x_{36}x_{38} + x_{35}x_{36} + x_{33} + x_{33}x_{35} + x_{32} + x_{32}x_{36} \\ &+ x_{31}x_{41}x_{42} + x_{31}x_{40}x_{41} + x_{31}x_{35}x_{37} + x_{30} + x_{29}x_{38} \\ &+ x_{29}x_{36}x_{37} + x_{29}x_{34}x_{37} + x_{29}x_{31} + x_{28} + x_{28}x_{42}x_{43} \\ &+ x_{28}x_{36}x_{38} + x_{28}x_{29} + x_{28}x_{29}x_{37} + x_{26} + x_{26}x_{29} + x_{25} \\ &+ x_{25}x_{28} + x_{24} + x_{24}x_{43} + x_{24}x_{32} + x_{24}x_{31} + x_{22} + x_{21} + x_{18} \end{aligned}$$

Here, $x = (x_1, x_2, \ldots, x_{128})$ represents the secret key ($x_i = k_i$). This recovered superpoly is considered a balanced Boolean function because it includes independent monomials x_{22}, x_{21}, and x_{18} that do not depend on other monomials.

Analyzing 194-Round Grain-128AEAD: Superpoly Recovery

The cube indices used for recovering the superpoly of the 194-round Grain-128AEAD are $\mathcal{I} = \{1, 2, \ldots, 67, 71, \ldots, 96\}$, with $IV_{68} = IV_{69} = IV_{70} = 0$. This resulted in identifying the

superpoly corresponding to $C_{\mathcal{I}}$, which consists of the sum of 38 monomials:

$$
\begin{aligned}
p(x) \;=\; & 1 + x_{43} + x_{42}x_{43} + x_{41} + x_{41}x_{42} + x_{40}x_{43} + x_{40}x_{42} + x_{40}x_{41} \\
& + x_{40}x_{41}x_{43} + x_{40}x_{41}x_{42} + x_{39} + x_{39}x_{41} + x_{39}x_{40} + x_{39}x_{40}x_{41}x_{42} \\
& + x_{38}x_{43} + x_{38}x_{41}x_{43} + x_{38}x_{41}x_{42} + x_{38}x_{40} + x_{38}x_{39} + x_{38}x_{39}x_{41}x_{42} \\
& + x_{37}x_{39}x_{40} + x_{37}x_{38} + x_{36}x_{40} + x_{35} + x_{35}x_{37}x_{41} + x_{35}x_{37}x_{40} \\
& + x_{34}x_{36} + x_{34}x_{35}x_{40} + x_{34}x_{35}x_{38} + x_{33} + x_{33}x_{42} + x_{33}x_{41}x_{43} \\
& + x_{32}x_{33} + x_{31} + x_{31}x_{42} + x_{31}x_{40} + x_{31}x_{39}x_{40} + x_{31}x_{36} \\
& + x_{31}x_{33} + x_{30}x_{43} + x_{30}x_{41}x_{42} + x_{30}x_{40}x_{42} + x_{30}x_{40}x_{41} \\
& + x_{30}x_{38} + x_{30}x_{37} + x_{30}x_{35}x_{37} + x_{30}x_{33} + x_{30}x_{33}x_{37} + x_{30}x_{32} \\
& + x_{30}x_{32}x_{41} + x_{30}x_{32}x_{40} + x_{30}x_{31} + x_{30}x_{31}x_{41} + x_{29} + x_{29}x_{41}x_{42} \\
& + x_{29}x_{40} + x_{29}x_{38} + x_{29}x_{35}x_{37}x_{41} + x_{29}x_{33} + x_{29}x_{32}x_{41} + x_{29}x_{30}x_{39} \\
& + x_{29}x_{30}x_{33} + x_{28} + x_{28}x_{41} + x_{28}x_{41}x_{43} + x_{28}x_{40} + x_{28}x_{35}x_{40} \\
& + x_{28}x_{35}x_{40}x_{41} + x_{28}x_{33} + x_{28}x_{31} + x_{28}x_{31}x_{40} + x_{28}x_{30} \\
& + x_{28}x_{30}x_{40} + x_{28}x_{30}x_{40} + x_{28}x_{30}x_{38} + x_{28}x_{30}x_{35}x_{40} + x_{28}x_{30}x_{31} \\
& + x_{28}x_{29}x_{39} + x_{27} + x_{27}x_{40} + x_{26}x_{40} + x_{26}x_{38} + x_{25}x_{40} + x_{25}x_{30}x_{40} \\
& + x_{24}x_{41}x_{43} + x_{24}x_{40} + x_{24}x_{40}x_{41} + x_{24}x_{38} + x_{24}x_{30}x_{41} + x_{24}x_{30}x_{40} \\
& + x_{24}x_{30}x_{39} + x_{24}x_{29}x_{41} + x_{24}x_{26} + x_{23} + x_{23}x_{40} + x_{23}x_{30} \\
& + x_{23}x_{29} + x_{22} + x_{22}x_{40} + x_{21} + x_{21}x_{38}x_{40} + x_{21}x_{33} \\
& + x_{21}x_{31}x_{40}x_{41} + x_{21}x_{31}x_{34}x_{36} + x_{21}x_{27} + x_{21}x_{26}x_{31}x_{40}x_{42} + x_{21}x_{26}x_{30} \\
& + x_{21}x_{26}x_{29}x_{31} + x_{21}x_{26}x_{28}x_{31} + x_{21}x_{23}x_{31} + x_{20}x_{42}x_{43} + x_{20}x_{40} \\
& + x_{20}x_{40}x_{42}x_{43} + x_{20}x_{38} + x_{20}x_{38}x_{40} + x_{20}x_{37}x_{39} + x_{20}x_{36} + x_{20}x_{36}x_{38} \\
& + x_{20}x_{35} + x_{20}x_{35}x_{36} + x_{20}x_{33} + x_{20}x_{33}x_{35}x_{36} + x_{20}x_{32} + x_{20}x_{32}x_{41}x_{42} \\
& + x_{20}x_{32} + x_{20}x_{32}x_{41}x_{42} + x_{20}x_{32}x_{35}x_{36} + x_{20}x_{29}x_{37}x_{41}x_{42} \\
& + x_{20}x_{29}x_{36}x_{38} + x_{20}x_{29}x_{36}x_{37} + x_{20}x_{29}x_{35}x_{37} + x_{20}x_{29}x_{31} \\
& + x_{20}x_{29}x_{30}x_{37} + x_{20}x_{29}x_{30}x_{32} + x_{20}x_{28} + x_{20}x_{28}x_{39}x_{40} + x_{20}x_{28}x_{37} \\
& + x_{20}x_{28}x_{33}x_{35} + x_{20}x_{28}x_{32} + x_{20}x_{28}x_{32}x_{35} + x_{20}x_{28}x_{29} + x_{20}x_{27} \\
& + x_{20}x_{26} + x_{20}x_{25} + x_{20}x_{25}x_{29} + x_{20}x_{24}x_{32} + x_{20}x_{24}x_{29}x_{37} \\
& + x_{20}x_{24}x_{28} + x_{20}x_{23} + x_{20}x_{23}x_{29}x_{32} + x_{20}x_{22}x_{28} + x_{20}x_{22}x_{28} \\
& + x_{20}x_{22}x_{24} + x_{20}x_{21} + x_{20}x_{21}x_{40}x_{42} + x_{20}x_{21}x_{23}x_{29} + x_{20}x_{21}x_{22} \\
& + x_{19}x_{20}x_{29} + x_{18}x_{20} + x_{17}x_{40} + x_{17}x_{38} + x_{13} + x_{11}x_{20}
\end{aligned}
$$

Here, $x = (x_1, x_2, \ldots, x_{128})$ represents the secret key ($x_i = k_i$). This recovered superpoly is a balanced Boolean function due to the inclusion of independent monomials x_{22}, x_{21}, and x_{18}.

Zero-Sum Distinguishers for 192-193 round Grain-128AEAD

To perform the cube attack on the 192-round Grain-128AEAD, we select cube indices of size 92:

$$\mathcal{I} = \{1, 2, \ldots, 66, 71, \ldots, 96\}$$

with the initialization vector conditions $IV_{67} = IV_{68} = IV_{69} = IV_{70} = 0$. In the weak-key scenario, the resulting superpoly does not depend on the secret key $x = (x_1, x_2, \ldots, x_{128})$. The attack on the 192-round Grain-128AEAD is a zero-sum distinguisher.

In a similar vein, in the cube attack on the 193-round Grain-128AEAD, we employ cube indices sized at 94:

$$\mathcal{I} = \{1, 2, \ldots, 68, 71, \ldots, 96\}$$

with $IV_{69} = IV_{70} = 0$. This represents the most extensive zero-sum distinguisher identified for Grain-128AEAD using division property-based cube attacks to date.

Table 6.4: Results of Superpoly Recovery for Different Cubes on Grain-128AEAD

Cube size	Cube variables	Round	Additional Conditional Key Variables
91	$IV \backslash \{s_{65}, s_{66}, s_{67}, s_{68}, s_{69}\}$	192	$b_{42}, b_{43}, b_{44}, b_{45}, b_{69}, b_{70}, b_{71}, b_{72}, b_{73}$ $b_{74}, b_{75}, b_{76}, b_{77}, b_{118}, b_{119}, b_{120}$ $b_{121}, b_{122}, b_{123}, b_{124}, b_{126}, b_{127}$
92	$IV \backslash \{s_{66}, s_{67}, s_{68}, s_{69}\}$	193	$b_{42}, b_{43}, b_{44}, b_{45}, b_{70}, b_{71}, b_{72}, b_{73}$ $b_{74}, b_{75}, b_{76}, b_{77}, b_{119}, b_{120}$ $b_{121}, b_{122}, b_{123}, b_{124}, b_{126}, b_{127}$
93	$IV \backslash \{s_{67}, s_{68}, s_{69}\}$	194	$b_{42}, b_{43}, b_{44}, b_{45}, b_{71}, b_{72}, b_{73}, b_{75}, b_{76}, b_{77}$ $b_{120}, b_{121}, b_{122}, b_{123}, b_{124}, b_{126}, b_{127}$
94	$IV \backslash \{s_{68}, s_{69}\}$	195	$b_{42}, b_{43}, b_{44}, b_{45}, b_{72}, b_{73}, b_{76}, b_{77}$ $b_{121}, b_{122}, b_{123}, b_{124}, b_{126}, b_{127}$

6.3 Conclusion

In this chapter, we re-examine cube attacks that utilize division properties, focusing on the NIST lightweight 3rd round candidate Grain-128AEAD. Initially, we identify effective cubes and develop an algorithm to determine conditional key bits for these cubes in Grain-128AEAD. By applying cube attacks based on the three-subset division property without an unknown subset, we successfully recover superpolies up to 195 rounds in the weak-key setting, representing the most advanced results on Grain-128AEAD to date. Additionally, we discover zero-sum distinguishers for 193-round Grain-128AEAD, marking the longest distinguisher achieved in this area.

Chapter 7

Conclusion

In this thesis, we present novel research contributions in the field of symmetric key cryptanalysis. We analyze the collision security of hash functions under the assumption that their underlying compression functions are collision-resistant. Furthermore, we propose innovative techniques to develop the most advanced automated models for conducting impossible differential attacks, zero correlation attacks, and integral distinguishing attacks on various block ciphers. Additionally, we make significant contributions to the division property-based cube attack. Our main results are included in Chapters 3, 4, 5, and 6. In the following, we give the concluding remarks and discuss potential future research directions.

7.1 Summary

In Chapter 3, we initially demonstrate the impossibility of proving collision security for the recently constructed hash modes ABR [9] and T_5 [8] based solely on the assumption of the collision security of the underlying compression function. Despite T_5 being proven collision-resistant under the random oracle model, we establish a lower bound on the number of calls to the underlying compression function required to preserve collision-resistant properties in a linear hash mode.

In Chapter 4, we extended the bit-wise CP model proposed in [13] to encompass ARX and AndRX designs. Furthermore, we demonstrate how this extended model can detect ZC distinguishers. Additionally, we introduce a novel CP model to identify ID distinguishers based on direct and indirect contradictions. This model is versatile and applicable beyond ARX and AndRX designs. Lastly, we integrate our new CP model for detecting ID distinguishers with the CP model used for key recovery, presenting a unified approach to identifying complete ID attacks.

In Chapter 5, we propose a method to model `BDPT` propagation in ciphers featuring binary (complex) linear layers. Additionally, we develop an efficient automatic search algorithm capable of accurately characterizing `BDPT` propagation. These advancements enable more precise `BDPT` analysis for ciphers incorporating complex linear layers, such as PRINCE, MANTIS, KLEIN, and PRIDE.

In Chapter 6, we revisit division property-based cube attacks and examine the NIST lightweight 3rd round candidate Grain-128AEAD in this context. Initially, we identify ef-

fective cubes and propose an algorithm to determine conditional key bits for these cubes in Grain-128AEAD. Subsequently, we apply the three-subset division property efficiently, utilizing cube attacks without unknown subset considerations, and successfully recover superpolies up to 195 rounds under weak-key conditions.

7.2 Future Direction

Building on the research presented in this thesis, several intriguing open problems can be identified for future investigation. These include:

- In Chapter 3, we have established a lower bound on the number of compression function calls required to preserve collision-resistant properties in a linear hash function structure. However, this achievement poses a challenging open problem: determining the lower bound on the compression function calls for a hash function employing nonlinear intermediate processing functions to maintain collision-resistant properties.

- In Chapter 4, we propose a unified CP model for constructing optimal ID attacks on AndRX ciphers. However, the challenge of developing a unified model to discover key-recovery attacks using ID distinguishers for ARX ciphers remains unresolved. Additionally, we introduce a CP model for constructing ZC distinguishers for any AND-RX cipher. Nevertheless, we encountered challenges in developing an automated satisfiability-based model for finding ZC distinguishers in ARX ciphers, which we identify as a focus for future work.

- In Chapter 5, we straightforwardly decompose the linear layer using the `COPY-XOR` technique for ciphers with non-binary linear layers, potentially overlooking balanced properties. Thus, accurately and efficiently modeling `BDPT` propagation for such ciphers remains an open problem. Furthermore, our current model utilizes a MILP solver, while SAT/SMT solvers are also widely recognized for their efficiency in this field. Exploring the implementation of our model with SAT/SMT solvers or similar approaches represents an intriguing avenue for future research.

- Cube attacks, particularly their extensions like dynamic cube attacks and their integration with other techniques, have garnered significant attention in cryptanalysis. They are highly relevant for stream ciphers and certain block cipher constructions. Finding cube variables (sets of input bits used to define a cube) for attacks is often done manually or semi-automatically. This limits the scalability and applicability of cube attacks to more complex ciphers. Developing automated tools for systematically identifying useful cube variables and configurations is a promising future direction. Machine learning-based techniques could also be explored to predict cube structures for a given cipher. Many modern cryptographic primitives follow the authenticated encryption (AEAD) paradigm (e.g., ASCON, Grain-128AEAD). Cube attacks have traditionally been applied to key-stream generators but have not been extensively studied for authenticated encryption schemes. It will be interesting to analyze how cube attacks, combined with side techniques (like the Division Property or statistical attacks), can exploit weaknesses in AEAD schemes.

Bibliography

[1] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, *Advanced encryption standard (aes)*, en, 2001-11-26 2001. DOI: https://doi.org/10.6028/NIST.FIPS.197.

[2] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard* (Information Security and Cryptography). Springer, 2002, ISBN: 3-540-42580-2. DOI: 10.1007/978-3-662-04722-4. URL: https://doi.org/10.1007/978-3-662-04722-4.

[3] M. Dworkin, *Sha-3 standard: Permutation-based hash and extendable-output functions*, en, 2015-08-04 2015. DOI: https://doi.org/10.6028/NIST.FIPS.202.

[4] *CAESAR competition*. DOI: https://competitions.cr.yp.to/caesar.html.

[5] *New european schemes for signatures, integrity, and encryption (NESSIE)*. DOI: https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development..

[6] *eSTREAM: The ECRYPT Stream Cipher Project*. DOI: https://competitions.cr.yp.to/estream.html.

[7] *National institute of standards and technology (nist): Lightweight cryptography standardization process*, 2019. DOI: https://csrc.nist.gov/projects/lightweight-cryptography.

[8] Y. Dodis, D. Khovratovich, N. Mouha, and M. Nandi, "T5: Hashing five inputs with three compression calls," in *2nd Conference on Information-Theoretic Cryptography, ITC 2021, July 23-26, 2021, Virtual Conference*, S. Tessaro, Ed., ser. LIPIcs, vol. 199, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 24:1–24:23. DOI: 10.4230/LIPICS.ITC.2021.24. URL: https://doi.org/10.4230/LIPIcs.ITC.2021.24.

[9] E. Andreeva, R. Bhattacharyya, and A. Roy, "Compactness of hashing modes and efficiency beyond merkle tree," in *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, A. Canteaut and F. Standaert, Eds., ser. Lecture Notes in Computer Science, vol. 12697, Springer, 2021, pp. 92–123. DOI: 10.1007/978-3-030-77886-6_4. URL: https://doi.org/10.1007/978-3-030-77886-6%5C_4.

[10] C. Dhar, Y. Dodis, and M. Nandi, "Revisiting Collision and Local Opening Analysis of ABR Hash," in *3rd Conference on Information-Theoretic Cryptography (ITC 2022)*, D. Dachman-Soled, Ed., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 230, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 11:1–11:22, ISBN: 978-3-95977-238-9. DOI: 10.4230/LIPIcs.ITC.2022.11. URL: https://drops.dagstuhl.de/opus/volltexte/2022/16489.

[11] D. Chakraborty and M. Nandi, "Lower bound on number of compression calls of a collision-resistance preserving hash," *IACR Commun. Cryptol.*, vol. 1, no. 3, p. 22, 2024. DOI: 10.62056/AKGY11ZN4. URL: https://doi.org/10.62056/akgy11zn4.

[12] H. Hadipour, S. Sadeghi, and M. Eichlseder, "Finding the impossible: Automated search for full impossible-differential, zero-correlation, and integral attacks," in *EUROCRYPT 2023*, ser. LNCS, vol. 14007, Springer, 2023, pp. 128–157. DOI: 10.1007/978-3-031-30634-1_5.

[13] H. Hadipour, S. Gerhalter, S. Sadeghi, and M. Eichlseder, "Improved search for integral, impossible differential and zero-correlation attacks application to ascon, forkskinny, skinny, mantis, PRESENT and qarmav2," *IACR Trans. Symmetric Cryptol.*, vol. 2024, no. 1, pp. 234–325, 2024. DOI: 10.46586/TOSC.V2024.I1.234-325.

[14] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The SIMON and SPECK lightweight block ciphers," in *Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, June 7-11, 2015*, ACM, 2015, 175:1–175:6. DOI: 10.1145/2744769.2747946. URL: https://doi.org/10.1145/2744769.2747946.

[15] D. J. Bernstein, *Chacha, a variant of salsa20*, 2008. URL: https://cr.yp.to/chacha.html.

[16] N. Mouha, B. Mennink, A. V. Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede, "Chaskey: An efficient MAC algorithm for 32-bit microcontrollers," in *SAC 2014*, A. Joux and A. M. Youssef, Eds., ser. LNCS, vol. 8781, Springer, 2014, pp. 306–323. DOI: 10.1007/978-3-319-13051-4_19.

[17] J. Aumasson and D. J. Bernstein, "SipHash: A fast short-input PRF," in *INDOCRYPT 2012*, S. D. Galbraith and M. Nandi, Eds., ser. LNCS, vol. 7668, Springer, 2012, pp. 489–508. DOI: 10.1007/978-3-642-34931-7_28.

[18] D. Hong, J.-K. Lee, D.-C. Kim, D. Kwon, K. H. Ryu, and D. Lee, "LEA: A 128-bit block cipher for fast encryption on common processors," in *WISA 2013*, ser. LNCS, vol. 8267, Springer, 2013, pp. 3–27. DOI: 10.1007/978-3-319-05149-9_1.

[19] G. Yang, B. Zhu, V. Suder, M. D. Aagaard, and G. Gong, "The simeck family of lightweight block ciphers," in *CHES 2015*, ser. LNCS, vol. 9293, Springer, 2015, pp. 307–329. DOI: 10.1007/978-3-662-48324-4_16.

[20] D. Chakraborty, H. Hadipour, P. H. Nguyen, and M. Eichlseder, "Finding complete impossible differential attacks on andrx ciphers and efficient distinguishers for ARX designs," *IACR Trans. Symmetric Cryptol.*, vol. 2024, no. 3, pp. 84–176, 2024. DOI: 10.46586/TOSC.V2024.I3.84-176. URL: https://doi.org/10.46586/tosc.v2024.i3.84-176.

[21] Y. Todo and M. Morii, "Bit-based division property and application to simon family," in *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, T. Peyrin, Ed., ser. Lecture Notes in Computer Science, vol. 9783, Springer, 2016, pp. 357–377. DOI: 10.1007/978-3-662-52993-5_18. URL: https://doi.org/10.1007/978-3-662-52993-5%5C_18.

[22] K. Hu and M. Wang, "Automatic search for a variant of division property using three subsets," in *Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings*, M. Matsui, Ed., ser. Lecture Notes in Computer Science, vol. 11405, Springer, 2019, pp. 412–432. DOI: 10.1007/978-3-030-12612-4_21. URL: https://doi.org/10.1007/978-3-030-12612-4%5C_21.

[23] S. Wang, B. Hu, J. Guan, K. Zhang, and T. Shi, "Milp-aided method of searching division property using three subsets and applications," in *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, S. D. Galbraith and S. Moriai, Eds., ser. Lecture Notes in Computer Science, vol. 11923, Springer, 2019, pp. 398–427. DOI: 10.1007/978-3-030-34618-8_14. URL: https://doi.org/10.1007/978-3-030-34618-8%5C_14.

[24] H. Liu, Z. Wang, and L. Zhang, *A model set method to search integral distinguishers based on division property for block ciphers*, Cryptology ePrint Archive, Paper 2022/720, https://eprint.iacr.org/2022/720, 2022. URL: https://eprint.iacr.org/2022/720.

[25] J. Borghoff, A. Canteaut, T. Güneysu, *et al.*, "PRINCE - A low-latency block cipher for pervasive computing applications (full version)," *IACR Cryptol. ePrint Arch.*, p. 529, 2012. URL: http://eprint.iacr.org/2012/529.

[26] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, M. Robshaw and J. Katz, Eds., ser. Lecture Notes in Computer Science, vol. 9815, Springer, 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5. URL: https://doi.org/10.1007/978-3-662-53008-5%5C_5.

[27] Z. Gong, S. Nikova, and Y. W. Law, "KLEIN: A new family of lightweight block ciphers," in *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, A. Juels and C. Paar, Eds., ser. Lecture Notes in Computer Science, vol. 7055, Springer, 2011, pp. 1–18. DOI: 10.1007/978-3-642-25286-0_1. URL: https://doi.org/10.1007/978-3-642-25286-0%5C_1.

[28] M. R. Albrecht, B. Driessen, E. B. Kavun, G. Leander, C. Paar, and T. Yalçin, "Block ciphers - focus on the linear layer (feat. PRIDE)," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, J. A. Garay and R. Gennaro, Eds., ser. Lecture

Notes in Computer Science, vol. 8616, Springer, 2014, pp. 57–76. DOI: 10.1007/978-3-662-44371-2_4. URL: https://doi.org/10.1007/978-3-662-44371-2%5C_4.

[29] S. Wang, B. Hu, J. Guan, K. Zhang, and T. Shi, "Exploring secret keys in searching integral distinguishers based on division property," *IACR Trans. Symmetric Cryptol.*, vol. 2020, no. 3, pp. 288–304, 2020. DOI: 10.13154/tosc.v2020.i3.288-304. URL: https://doi.org/10.13154/tosc.v2020.i3.288-304.

[30] D. Chakraborty, "Finding three-subset division property for ciphers with complex linear layers," in *Progress in Cryptology - INDOCRYPT 2022 - 23rd International Conference on Cryptology in India, Kolkata, India, December 11-14, 2022, Proceedings*, T. Isobe and S. Sarkar, Eds., ser. Lecture Notes in Computer Science, vol. 13774, Springer, 2022, pp. 398–421. DOI: 10.1007/978-3-031-22912-1_18. URL: https://doi.org/10.1007/978-3-031-22912-1%5C_18.

[31] I. Dinur and A. Shamir, "Cube attacks on tweakable black box polynomials," in *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, A. Joux, Ed., ser. Lecture Notes in Computer Science, vol. 5479, Springer, 2009, pp. 278–299. DOI: 10.1007/978-3-642-01001-9_16. URL: https://doi.org/10.1007/978-3-642-01001-9%5C_16.

[32] M. Hell, T. Johansson, W. Meier, J. Sönnerup, and H. Yoshida, "An AEAD variant of the grain stream cipher," in *Codes, Cryptology and Information Security - Third International Conference, C2SI 2019, Rabat, Morocco, April 22-24, 2019, Proceedings - In Honor of Said El Hajji*, C. Carlet, S. Guilley, A. Nitaj, and E. M. Souidi, Eds., ser. Lecture Notes in Computer Science, vol. 11445, Springer, 2019, pp. 55–71. DOI: 10.1007/978-3-030-16458-4_5. URL: https://doi.org/10.1007/978-3-030-16458-4%5C_5.

[33] Y. Hao, G. Leander, W. Meier, Y. Todo, and Q. Wang, "Modeling for three-subset division property without unknown subset - improved cube attacks against trivium and grain-128aead," in *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, A. Canteaut and Y. Ishai, Eds., ser. Lecture Notes in Computer Science, vol. 12105, Springer, 2020, pp. 466–495. DOI: 10.1007/978-3-030-45721-1_17. URL: https://doi.org/10.1007/978-3-030-45721-1%5C_17.

[34] K. Hu, S. Sun, Y. Todo, M. Wang, and Q. Wang, "Massive superpoly recovery with nested monomial predictions," in *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, M. Tibouchi and H. Wang, Eds., ser. Lecture Notes in Computer Science, vol. 13090, Springer, 2021, pp. 392–421. DOI: 10.1007/978-3-030-92062-3_14. URL: https://doi.org/10.1007/978-3-030-92062-3%5C_14.

[35] J. He, K. Hu, B. Preneel, and M. Wang, "Stretching cube attacks: Improved methods to recover massive superpolies," *IACR Cryptol. ePrint Arch.*, p. 1218, 2022. URL: https://eprint.iacr.org/2022/1218.

[36] D. Chakraborty and S. Pal, "Superpoly recovery of grain-128aead using division property," in *Innovative Security Solutions for Information Technology and Communications - 15th International Conference, SecITC 2022, Virtual Event, December 8-9, 2022, Revised Selected Papers*, G. Bella, M. Doinea, and H. Janicke, Eds., ser. Lecture Notes in Computer Science, vol. 13809, Springer, 2022, pp. 65–80. DOI: 10.1007/978-3-031-32636-3_4. URL: https://doi.org/10.1007/978-3-031-32636-3%5C_4.

[37] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "PRESENT: an ultra-lightweight block cipher," in *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, P. Paillier and I. Verbauwhede, Eds., ser. Lecture Notes in Computer Science, vol. 4727, Springer, 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31. URL: https://doi.org/10.1007/978-3-540-74735-2%5C_31.

[38] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo, "GIFT: A small present - towards reaching the limit of lightweight encryption," in *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, W. Fischer and N. Homma, Eds., ser. Lecture Notes in Computer Science, vol. 10529, Springer, 2017, pp. 321–345. DOI: 10.1007/978-3-319-66787-4_16. URL: https://doi.org/10.1007/978-3-319-66787-4%5C_16.

[39] National Institute of Standards and Technology, *Data encryption standard (des)*, FIPS Publication 46-3, Oct. 1999.

[40] K. Sakiyama, Y. Sasaki, and Y. Li, *Security of Block Ciphers: From Algorithm Design to Hardware Implementation*, 1st. Wiley Publishing, 2015, ISBN: 1118660013.

[41] E. Biham and A. Shamir, "Differential cryptanalysis of des-like cryptosystems," *J. Cryptol.*, vol. 4, no. 1, pp. 3–72, 1991. DOI: 10.1007/BF00630563. URL: https://doi.org/10.1007/BF00630563.

[42] L. Knudsen, "Deal-a 128-bit block cipher," *Complexity*, vol. 258, no. 2, 1998.

[43] E. Biham, A. Biryukov, and A. Shamir, "Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 12–23, ISBN: 978-3-540-48910-8.

[44] C. Boura, M. Naya-Plasencia, and V. Suder, "Scrutinizing and improving impossible differential attacks: Applications to CLEFIA, Camellia, LBlock and Simon," in *ASIACRYPT 2014*, ser. LNCS, Springer, vol. 8873, 2014, pp. 179–199. DOI: 10.1007/978-3-662-45611-8_10.

[45] J. Lu, J. Kim, N. Keller, and O. Dunkelman, "Improving the efficiency of impossible differential cryptanalysis of reduced Camellia and MISTY1," in *CT-RSA 2008*, ser. LNCS, vol. 4964, Springer, 2008, pp. 370–386. DOI: 10.1007/978-3-540-79263-5_24.

[46] M. Matsui, "Linear cryptanalysis method for DES cipher," in *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, T. Helleseth, Ed., ser. Lecture Notes in Computer Science, vol. 765, Springer, 1993, pp. 386–397. DOI: 10.1007/3-540-48285-7_33. URL: https://doi.org/10.1007/3-540-48285-7%5C_33.

[47] A. Tardy-Corfdir and H. Gilbert, "A known plaintext attack of FEAL-4 and FEAL-6," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, J. Feigenbaum, Ed., ser. Lecture Notes in Computer Science, vol. 576, Springer, 1991, pp. 172–181. DOI: 10.1007/3-540-46766-1_12. URL: https://doi.org/10.1007/3-540-46766-1%5C_12.

[48] J. Daemen, R. Govaerts, and J. Vandewalle, "Correlation matrices," in *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, B. Preneel, Ed., ser. Lecture Notes in Computer Science, vol. 1008, Springer, 1994, pp. 275–285. DOI: 10.1007/3-540-60590-8_21. URL: https://doi.org/10.1007/3-540-60590-8%5C_21.

[49] A. Bogdanov and V. Rijmen, "Linear hulls with correlation zero and linear cryptanalysis of block ciphers," *Des. Codes Cryptogr.*, vol. 70, no. 3, pp. 369–383, 2014. DOI: 10.1007/s10623-012-9697-z.

[50] A. Bogdanov and M. Wang, "Zero correlation linear cryptanalysis with reduced data complexity," in *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, A. Canteaut, Ed., ser. Lecture Notes in Computer Science, vol. 7549, Springer, 2012, pp. 29–48. DOI: 10.1007/978-3-642-34047-5_3. URL: https://doi.org/10.1007/978-3-642-34047-5%5C_3.

[51] A. Bogdanov, G. Leander, K. Nyberg, and M. Wang, "Integral and multidimensional linear distinguishers with correlation zero," in *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, X. Wang and K. Sako, Eds., ser. Lecture Notes in Computer Science, vol. 7658, Springer, 2012, pp. 244–261. DOI: 10.1007/978-3-642-34961-4_16. URL: https://doi.org/10.1007/978-3-642-34961-4%5C_16.

[52] H. Soleimany and K. Nyberg, "Zero-correlation linear cryptanalysis of reduced-round lblock," *Des. Codes Cryptogr.*, vol. 73, no. 2, pp. 683–698, 2014. DOI: 10.1007/S10623-014-9976-Y. URL: https://doi.org/10.1007/s10623-014-9976-y.

[53] J. Daemen, L. R. Knudsen, and V. Rijmen, "The block cipher square," in *Fast Software Encryption, 4th International Workshop, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings*, E. Biham, Ed., ser. Lecture Notes in Computer Science, vol. 1267, Springer, 1997, pp. 149–165. DOI: 10.1007/BFb0052343. URL: https://doi.org/10.1007/BFb0052343.

[54] Y. Todo, "Structural evaluation by generalized integral property," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, E. Oswald and M. Fischlin, Eds., ser. Lecture Notes in Computer Science, vol. 9056, Springer, 2015, pp. 287–314. DOI: 10.1007/978-3-662-46800-5_12. URL: https://doi.org/10.1007/978-3-662-46800-5%5C_12.

[55] M. Ågren, M. Hell, T. Johansson, and W. Meier, "Grain-128a: A new version of grain-128 with optional authentication," *Int. J. Wirel. Mob. Comput.*, vol. 5, no. 1, pp. 48–59, 2011. DOI: 10.1504/IJWMC.2011.044106. URL: https://doi.org/10.1504/IJWMC.2011.044106.

[56] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "Sponge functions," in *ECRYPT hash workshop*, vol. 200, 2007, p. 21.

[57] J. Bierbrauer, T. Johansson, G. Kabatianskii, and B. J. M. Smeets, "On families of hash functions via geometric codes and concatenation," in *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, D. R. Stinson, Ed., ser. Lecture Notes in Computer Science, vol. 773, Springer, 1993, pp. 331–342. DOI: 10.1007/3-540-48329-2_28. URL: https://doi.org/10.1007/3-540-48329-2%5C_28.

[58] J. Black, P. Rogaway, and T. Shrimpton, "Black-box analysis of the block-cipher-based hash-function constructions from PGV," in *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, M. Yung, Ed., ser. Lecture Notes in Computer Science, vol. 2442, Springer, 2002, pp. 320–335. DOI: 10.1007/3-540-45708-9_21. URL: https://doi.org/10.1007/3-540-45708-9%5C_21.

[59] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Duplexing the sponge: Single-pass authenticated encryption and other applications," in *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*, A. Miri and S. Vaudenay, Eds., ser. Lecture Notes in Computer Science, vol. 7118, Springer, 2011, pp. 320–337. DOI: 10.1007/978-3-642-28496-0_19. URL: https://doi.org/10.1007/978-3-642-28496-0%5C_19.

[60] P. Rogaway and J. P. Steinberger, "Constructing cryptographic hash functions from fixed-key blockciphers," in *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, D. A. Wagner, Ed., ser. Lecture Notes in Computer Science, vol. 5157, Springer, 2008, pp. 433–450. DOI: 10.1007/978-3-540-85174-5_24. URL: https://doi.org/10.1007/978-3-540-85174-5%5C_24.

[61] T. Ristenpart and T. Shrimpton, "How to build a hash function from any collision-resistant function," in *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, K. Kurosawa, Ed., ser. Lecture Notes in Computer Science, vol. 4833, Springer, 2007, pp. 147–163. DOI: 10.1007/978-3-540-76900-2_9. URL: https://doi.org/10.1007/978-3-540-76900-2%5C_9.

[62] J. Black, M. Cochran, and T. Shrimpton, "On the impossibility of highly-efficient blockcipher-based hash functions," *J. Cryptol.*, vol. 22, no. 3, pp. 311–329, 2009. DOI: 10.1007/s00145-008-9030-1. URL: https://doi.org/10.1007/s00145-008-9030-1.

[63] I. Damgård, "A design principle for hash functions," in *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, G. Brassard, Ed., ser. Lecture Notes in Computer Science, vol. 435, Springer, 1989, pp. 416–427. DOI: 10.1007/0-387-34805-0_39. URL: https://doi.org/10.1007/0-387-34805-0%5C_39.

[64] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, G. Brassard, Ed., ser. Lecture Notes in Computer Science, vol. 435, Springer, 1989, pp. 218–238. DOI: 10.1007/0-387-34805-0_21. URL: https://doi.org/10.1007/0-387-34805-0%5C_21.

[65] R. C. Merkle, "Protocols for public key cryptosystems," in *Proceedings of the 1980 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 14-16, 1980*, IEEE Computer Society, 1980, pp. 122–134. DOI: 10.1109/SP.1980.10006. URL: https://doi.org/10.1109/SP.1980.10006.

[66] D. Benjamin, "Batch Signing for TLS," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-batch-signing-00, Jan. 2020, Work in Progress, 10 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tls-batch-signing-00.

[67] J. Buchmann, L. C. C. García, E. Dahmen, M. Döring, and E. Klintsevich, "Cmss – an improved merkle signature scheme," in *Progress in Cryptology - INDOCRYPT 2006*, R. Barua and T. Lange, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 349–363, ISBN: 978-3-540-49769-1.

[68] J. Buchmann, E. Dahmen, E. Klintsevich, K. Okeya, and C. Vuillaume, "Merkle signatures with virtually unlimited signature capacity," in *Proceedings of the 5th International Conference on Applied Cryptography and Network Security*, ser. ACNS '07, Zhuhai, China: Springer-Verlag, 2007, pp. 31–45, ISBN: 9783540727378. DOI: 10.1007/978-3-540-72738-5_3. URL: https://doi.org/10.1007/978-3-540-72738-5_3.

[69] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The sphincs$^+$ signature framework," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, L. Cavallaro, J. Kinder, X. Wang, and J. Katz, Eds., ACM, 2019, pp. 2129–2146. DOI: 10.1145/3319535.3363229. URL: https://doi.org/10.1145/3319535.3363229.

[70] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," *J. Cryptol.*, vol. 3, no. 2, pp. 99–111, 1991. DOI: 10.1007/BF00196791. URL: https://doi.org/10.1007/BF00196791.

[71] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, T. Johansson and P. Q. Nguyen, Eds., ser. Lecture Notes in Computer Science, vol. 7881, Springer, 2013, pp. 626–645. DOI: 10.1007/978-3-642-38348-9_37. URL: https://doi.org/10.1007/978-3-642-38348-9%5C_37.

[72] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von neumann architecture," in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, K. Fu and J. Jung, Eds., USENIX Association, 2014, pp. 781–796. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson.

[73] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," *IACR Cryptol. ePrint Arch.*, p. 349, 2014. URL: http://eprint.iacr.org/2014/349.

[74] M. O. Rabin, "Digitalized signatures and public-key functions as intractable as factorization," USA, Tech. Rep., 1979.

[75] R. M.O., "Digitalized signatures," *Foundations of Secure Computation*, pp. 155–168, 1978. URL: https://cir.nii.ac.jp/crid/1573105973881933824.

[76] M. Stam, "Beyond uniformity: Better security/efficiency tradeoffs for compression functions," in *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, D. A. Wagner, Ed., ser. Lecture Notes in Computer Science, vol. 5157, Springer, 2008, pp. 397–412. DOI: 10.1007/978-3-540-85174-5_22. URL: https://doi.org/10.1007/978-3-540-85174-5%5C_22.

[77] J. P. Steinberger, "Stam's collision resistance conjecture," in *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, H. Gilbert, Ed., ser. Lecture Notes in Computer Science, vol. 6110, Springer, 2010, pp. 597–615. DOI: 10.1007/978-3-642-13190-5_30. URL: https://doi.org/10.1007/978-3-642-13190-5%5C_30.

[78] J. P. Steinberger, X. Sun, and Z. Yang, "Stam's conjecture and threshold phenomena in collision resistance," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, R. Safavi-Naini and R. Canetti, Eds., ser. Lecture Notes in Computer Science, vol. 7417, Springer, 2012, pp. 384–405. DOI: 10.1007/978-3-642-32009-5_23. URL: https://doi.org/10.1007/978-3-642-32009-5%5C_23.

[79] T. Shrimpton and M. Stam, "Building a collision-resistant compression function from non-compressing primitives," in *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, L. Aceto, I. Damgård, L. A. Goldberg, M. M.

Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds., ser. Lecture Notes in Computer Science, vol. 5126, Springer, 2008, pp. 643–654. DOI: 10.1007/978-3-540-70583-3_52. URL: https://doi.org/10.1007/978-3-540-70583-3%5C_52.

[80] P. Rogaway, "Formalizing human ignorance," in *Progressin Cryptology - VIETCRYPT 2006, First International Conference on Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, P. Q. Nguyen, Ed., ser. Lecture Notes in Computer Science, vol. 4341, Springer, 2006, pp. 211–228. DOI: 10.1007/11958239_14. URL: https://doi.org/10.1007/11958239%5C_14.

[81] E. H. Bareiss, "Sylvester's identity and multistep integer-preserving gaussian elimination," *Mathematics of Computation*, vol. 22, no. 103, pp. 565–578, 1968, ISSN: 00255718, 10886842. URL: http://www.jstor.org/stable/2004533 (visited on 11/20/2023).

[82] E. Biham, A. Biryukov, and A. Shamir, "Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials," in *EUROCRYPT 1999*, ser. LNCS, vol. 1592, Springer, 1999, pp. 12–23. DOI: 10.1007/3-540-48910-X_2.

[83] L. Knudsen, "DEAL – a 128-bit block cipher," *Complexity*, vol. 258, no. 2, p. 216, 1998.

[84] Y. Liu, L. Li, D. Gu, X. Wang, Z. Liu, J. Chen, and W. Li, "New observations on impossible differential cryptanalysis of reduced-round camellia," in *FSE 2012*, A. Canteaut, Ed., ser. LNCS, vol. 7549, Springer, 2012, pp. 90–109. DOI: 10.1007/978-3-642-34047-5_6.

[85] W. Zhang, W. Wu, and D. Feng, "New results on impossible differential cryptanalysis of reduced AES," in *ICISC 2007*, K.-H. Nam and G. Rhee, Eds., ser. LNCS, vol. 4817, Springer, 2007, pp. 239–250. DOI: 10.1007/978-3-540-76788-6_19.

[86] J. Lu, O. Dunkelman, N. Keller, and J. Kim, "New impossible differential attacks on AES," in *INDOCRYPT 2008*, ser. LNCS, vol. 5365, Springer, 2008, pp. 279–293. DOI: 10.1007/978-3-540-89754-5_22.

[87] H. Mala, M. Dakhilalian, V. Rijmen, and M. Modarres-Hashemi, "Improved impossible differential cryptanalysis of 7-round AES-128," in *INDOCRYPT 2010*, G. Gong and K. C. Gupta, Eds., ser. LNCS, vol. 6498, Springer, 2010, pp. 282–291. DOI: 10.1007/978-3-642-17401-8_20.

[88] J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999.

[89] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-bit blockcipher CLEFIA (extended abstract)," in *FSE 2007*, ser. LNCS, vol. 4593, Springer, 2007, pp. 181–195. DOI: 10.1007/978-3-540-74619-5_12.

[90] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *CRYPTO 2016*, Springer, 2016, pp. 123–153. DOI: 10.1007/978-3-662-53008-5_5.

[91] J. Kim, S. Hong, J. Sung, C. Lee, and S. Lee, "Impossible differential cryptanalysis for block cipher structures," in *INDOCRYPT 2003*, T. Johansson and S. Maitra, Eds., ser. LNCS, vol. 2904, Springer, 2003, pp. 82–96. DOI: 10.1007/978-3-540-24582-7_6.

[92] Y. Luo, X. Lai, Z. Wu, and G. Gong, "A unified method for finding impossible differentials of block cipher structures," *Inf. Sci.*, vol. 263, pp. 211–220, 2014, See also http://eprint.iacr.org/2009/627. DOI: 10.1016/J.INS.2013.08.051.

[93] P. Derbez and P.-A. Fouque, "Automatic search of meet-in-the-middle and impossible differential attacks," in *CRYPTO 2016*, ser. LNCS, vol. 9815, Springer, 2016, pp. 157–184. DOI: 10.1007/978-3-662-53008-5_6.

[94] Y. Sasaki and Y. Todo, "New impossible differential search tool from design and cryptanalysis aspects," in *EUROCRYPT 2017*, Cham: Springer, 2017, pp. 185–215. DOI: 10.1007/978-3-319-56617-7_7.

[95] T. Cui, S. Chen, K. Jia, K. Fu, and M. Wang, "New automatic search tool for impossible differentials and zero-correlation linear approximations," *Sci. China Inf. Sci.*, vol. 64, no. 2, 2021, See also https://eprint.iacr.org/2016/689. DOI: 10.1007/S11432-018-1506-4.

[96] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "Ascon v1.2: Lightweight authenticated encryption and hashing," *Journal of Cryptology*, vol. 34, no. 3, p. 33, 2021. DOI: 10.1007/s00145-021-09398-9.

[97] S. Sadeghi and N. Bagheri, "Improved zero-correlation and impossible differential cryptanalysis of reduced-round SIMECK block cipher," *IET Inf. Secur.*, vol. 12, no. 4, pp. 314–325, 2018. DOI: 10.1049/IET-IFS.2016.0590.

[98] J. Ren and S. Chen, "Cryptanalysis of reduced-round speck," *IEEE Access*, vol. 7, pp. 63 045–63 056, 2019. DOI: 10.1109/ACCESS.2019.2917015.

[99] H. Lee, H. Kang, D. Hong, J. Sung, and S. Hong, "New impossible differential characteristic of SPECK64 using MILP," *IACR Cryptol. ePrint Arch.*, p. 1137, 2016. URL: http://eprint.iacr.org/2016/1137.

[100] M. Saberi, N. Bagheri, and S. Sadeghi, "Impossible differential and zero-correlation linear cryptanalysis of marx, marx2, chaskey and speck32," in *ICCKE 2021*, 2021, pp. 48–54. DOI: 10.1109/ICCKE54056.2021.9721479.

[101] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "Minizinc: Towards a standard CP modelling language," in *CP 2007*, C. Bessiere, Ed., ser. LNCS, vol. 4741, Springer, 2007, pp. 529–543. DOI: 10.1007/978-3-540-74970-7_38.

[102] Z. Chen, N. Wang, and X. Wang, "Impossible differential cryptanalysis of reduced round SIMON," *IACR Cryptol. ePrint Arch.*, p. 286, 2015. URL: http://eprint.iacr.org/2015/286.

[103] K. Zhang, X. Lai, L. Wang, J. Guan, and B. Hu, "A revisited security evaluation of simeck family ciphers against impossible differential cryptanalysis," *Sci. China Inf. Sci.*, vol. 66, no. 3, 2023. DOI: 10.1007/S11432-022-3466-X.

[104] K. Qiao, L. Hu, and S. Sun, "Differential analysis on simeck and SIMON with dynamic key-guessing techniques," in *ICISSP 2016*, O. Camp, S. Furnell, and P. Mori, Eds., ser. Communications in Computer and Information Science, vol. 691, Springer, 2016, pp. 64–85. DOI: 10.1007/978-3-319-54433-5_5.

[105] H. Chen and X. Wang, "Improved linear hull attack on round-reduced simon with dynamic key-guessing techniques," in *FSE 2016*, T. Peyrin, Ed., ser. LNCS, vol. 9783, Springer, 2016, pp. 428–449. DOI: 10.1007/978-3-662-52993-5_22.

[106] L. Song, L. Hu, B. Ma, and D. Shi, "Match box meet-in-the-middle attacks on the SIMON family of block ciphers," in *LightSec 2014*, T. Eisenbarth and E. Öztürk, Eds., ser. LNCS, vol. 8898, Springer, 2014, pp. 140–151. DOI: 10.1007/978-3-319-16363-5_9.

[107] Y. Lv, D. Shi, Y. Guo, Q. Chen, L. Hu, and Z. Guo, "Automatic Demirci-Selçuk meet-in-the-middle attack on SIMON," *Comput. J.*, vol. 66, no. 12, pp. 3052–3068, 2023. DOI: 10.1093/COMJNL/BXAC149.

[108] L. Sun, K. Fu, and M. Wang, "Improved zero-correlation cryptanalysis on SIMON," in *Inscrypt*, ser. LNCS, vol. 9589, Springer, 2015, pp. 125–143.

[109] Z. Chu, H. Chen, X. Wang, X. Dong, and L. Li, "Improved integral attacks on SIMON32 and SIMON48 with dynamic key-guessing techniques," *Secur. Commun. Networks*, vol. 2018, 5160237:1–5160237:11, 2018. DOI: 10.1155/2018/5160237.

[110] N. Wang, X. Wang, K. Jia, and J. Zhao, "Differential attacks on reduced SIMON versions with dynamic key-guessing techniques," *Sci. China Inf. Sci.*, vol. 61, no. 9, 098103:1–098103:3, 2018. DOI: 10.1007/S11432-017-9231-5.

[111] G. Leurent, C. Pernot, and A. Schrottenloher, "Clustering effect in simon and simeck," in *ASIACRYPT 2021*, M. Tibouchi and H. Wang, Eds., ser. LNCS, vol. 13090, Springer, 2021, pp. 272–302. DOI: 10.1007/978-3-030-92062-3_10.

[112] L. Qin, H. Chen, and X. Wang, "Linear hull attack on round-reduced simeck with dynamic key-guessing techniques," in *Information Security and Privacy - 21st Australasian Conference, ACISP 2016, Proceedings, Part II*, ser. LNCS, vol. 9723, Springer, 2016, pp. 409–424. DOI: 10.1007/978-3-319-40367-0_26.

[113] K. Zhang, J. Guan, B. Hu, and D. Lin, "Security evaluation on simeck against zero-correlation linear cryptanalysis," *IET Inf. Secur.*, vol. 12, no. 1, pp. 87–93, 2018. DOI: 10.1049/IET-IFS.2016.0503.

[114] H. Li, J. Ren, and S. Chen, "Improved integral attack on reduced-round simeck," *IEEE Access*, vol. 7, pp. 118 806–118 814, 2019. DOI: 10.1109/ACCESS.2019.2936834.

[115] H. Hadipour, P. Derbez, and M. Eichlseder, *Revisiting differential-linear attacks via a boomerang perspective with application to aes, ascon, clefia, skinny, present, knot, twine, warp, lblock, simeck, and serpent*, IACR Cryptology ePrint Archive, Report 2016/689, 2024. URL: https://eprint.iacr.org/2024/255.

[116] H. Hadipour, M. Nageler, and M. Eichlseder, "Throwing boomerangs into Feistel structures: Application to CLEFIA, WARP, LBlock, LBlock-s and TWINE," *IACR Trans. Symmetric Cryptol.*, vol. 2022, no. 3, pp. 271–302, 2022. DOI: 10.46586/tosc.v2022.i3.271-302.

[117] H. Hadipour and M. Eichlseder, "Autoguess: A tool for finding guess-and-determine attacks and key bridges," in *ACNS 2022*, G. Ateniese and D. Venturi, Eds., ser. LNCS, vol. 13269, Springer, 2022, pp. 230–250. DOI: 10.1007/978-3-031-09234-3_12.

[118] T. Peyrin and Q. Q. Tan, "Mind your path: On (key) dependencies in differential characteristics," *IACR Trans. Symmetric Cryptol.*, vol. 2022, no. 4, pp. 179–207, 2022. DOI: 10.46586/TOSC.V2022.I4.179-207.

[119] T. Isobe and K. Shibutani, "Generic key recovery attack on feistel scheme," LNCS, vol. 8269, pp. 464–485, 2013. DOI: 10.1007/978-3-642-42033-7_24.

[120] Z. Liu, Y. Li, L. Jiao, and M. Wang, "A new method for searching optimal differential and linear trails in ARX ciphers," *IEEE Trans. Inf. Theory*, vol. 67, no. 2, pp. 1054–1068, 2021. DOI: 10.1109/TIT.2020.3040543.

[121] F. Wang and G. Wang, "Improved differential-linear attack with application to round-reduced speck32/64," in *ACNS 2022*, ser. LNCS, vol. 13269, Springer, 2022, pp. 792–808. DOI: 10.1007/978-3-031-09234-3_39.

[122] A. Biryukov, L. C. dos Santos, J. S. Teh, A. Udovenko, and V. Velichkov, "Meet-in-the-filter and dynamic counting with applications to speck," in *ACNS 2023*, ser. LNCS, vol. 13905, Springer, 2023, pp. 149–177. DOI: 10.1007/978-3-031-33488-7_6.

[123] J. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger, "New features of latin dances: Analysis of salsa, chacha, and rumba," in *FSE 2008*, K. Nyberg, Ed., ser. LNCS, vol. 5086, Springer, 2008, pp. 470–488. DOI: 10.1007/978-3-540-71039-4_30.

[124] E. Bellini, D. Gérault, J. Grados, R. H. Makarim, and T. Peyrin, "Boosting differential-linear cryptanalysis of chacha7 with MILP," *IACR Trans. Symmetric Cryptol.*, vol. 2023, no. 2, pp. 189–223, 2023. DOI: 10.46586/TOSC.V2023.I2.189-223.

[125] L. Kraleva, T. Ashur, and V. Rijmen, "Rotational cryptanalysis on MAC algorithm chaskey," in *ACNS 2020*, ser. LNCS, vol. 12146, Springer, 2020, pp. 153–168. DOI: 10.1007/978-3-030-57808-4_8.

[126] G. Leurent, "Improved differential-linear cryptanalysis of 7-round chaskey with partitioning," in *EUROCRYPT 2016*, M. Fischlin and J. Coron, Eds., ser. LNCS, vol. 9665, Springer, 2016, pp. 344–371. DOI: 10.1007/978-3-662-49890-3_14.

[127] K. Zhang, J. Guan, and B. Hu, "Zero correlation linear cryptanalysis on LEA family ciphers," *J. Commun.*, vol. 11, no. 7, pp. 677–685, 2016. DOI: 10.12720/JCM.11.7.677-685.

[128] C. Dobraunig, F. Mendel, and M. Schläffer, "Differential cryptanalysis of SipHash," in *SAC 2014*, A. Joux and A. M. Youssef, Eds., ser. LNCS, vol. 8781, Springer, 2014, pp. 165–182. DOI: 10.1007/978-3-319-13051-4_10.

[129] W. Xin, Y. Liu, B. Sun, and C. Li, "Improved cryptanalysis on SipHash," in *CANS 2019*, Y. Mu, R. H. Deng, and X. Huang, Eds., ser. LNCS, vol. 11829, Springer, 2019, pp. 61–79. DOI: 10.1007/978-3-030-31578-8_4.

[130] L. He and H. Yu, "Cryptanalysis of reduced-round SipHash," *Comput. J.*, vol. 67, no. 3, pp. 875–883, 2024. DOI: 10.1093/COMJNL/BXAD026.

[131] K. Zhang, S. Wang, X. Lai, L. Wang, J. Guan, B. Hu, and T. Shi, "Impossible differential cryptanalysis and a security evaluation framework for and-rx ciphers," *IEEE Transactions on Information Theory*, pp. 1–1, 2023. DOI: 10.1109/TIT.2023.3292241.

[132] C. Boura and A. Canteaut, "Another view of the division property," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, M. Robshaw and J. Katz, Eds., ser. Lecture Notes in Computer Science, vol. 9814, Springer, 2016, pp. 654–682. DOI: 10.1007/978-3-662-53018-4_24. URL: https://doi.org/10.1007/978-3-662-53018-4%5C_24.

[133] Z. Xiang, W. Zhang, Z. Bao, and D. Lin, "Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers," in *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, J. H. Cheon and T. Takagi, Eds., ser. Lecture Notes in Computer Science, vol. 10031, 2016, pp. 648–678. DOI: 10.1007/978-3-662-53887-6_24. URL: https://doi.org/10.1007/978-3-662-53887-6%5C_24.

[134] L. Sun, W. Wang, and M. Wang, "Milp-aided bit-based division property for primitives with non-bit-permutation linear layers," *IACR Cryptol. ePrint Arch.*, p. 811, 2016. URL: http://eprint.iacr.org/2016/811.

[135] L. Sun, W. Wang, and M. Wang, "Automatic search of bit-based division property for ARX ciphers and word-based division property," in *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, T. Takagi and T. Peyrin, Eds., ser. Lecture Notes in Computer Science, vol. 10624, Springer, 2017, pp. 128–157. DOI: 10.1007/978-3-319-70694-8_5. URL: https://doi.org/10.1007/978-3-319-70694-8%5C_5.

[136] W. Zhang and V. Rijmen, "Division cryptanalysis of block ciphers with a binary diffusion layer," *IET Inf. Secur.*, vol. 13, no. 2, pp. 87–95, 2019. DOI: 10.1049/iet-ifs.2018.5151. URL: https://doi.org/10.1049/iet-ifs.2018.5151.

[137] K. Hu, Q. Wang, and M. Wang, "Finding bit-based division property for ciphers with complex linear layer," *IACR Cryptol. ePrint Arch.*, p. 547, 2020. URL: https://eprint.iacr.org/2020/547.

[138] P. Hebborn, B. Lambin, G. Leander, and Y. Todo, "Lower bounds on the degree of block ciphers," in *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, S. Moriai and H. Wang, Eds., ser. Lecture Notes in Computer Science, vol. 12491, Springer, 2020, pp. 537–566. DOI: 10.1007/978-3-030-64837-4_18. URL: https://doi.org/10.1007/978-3-030-64837-4%5C_18.

[139] B. Lambin, P. Derbez, and P. Fouque, "Linearly equivalent s-boxes and the division property," *Des. Codes Cryptogr.*, vol. 88, no. 10, pp. 2207–2231, 2020. DOI: 10.1007/s10623-020-00773-4. URL: https://doi.org/10.1007/s10623-020-00773-4.

[140] P. Hebborn, B. Lambin, G. Leander, and Y. Todo, "Strong and tight security guarantees against integral distinguishers," in *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, M. Tibouchi and H. Wang, Eds., ser. Lecture Notes in Computer Science, vol. 13090, Springer, 2021, pp. 362–391. DOI: 10.1007/978-3-030-92062-3_13. URL: https://doi.org/10.1007/978-3-030-92062-3%5C_13.

[141] S. Kölbl, G. Leander, and T. Tiessen, "Observations on the SIMON block cipher family," in *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, R. Gennaro and M. Robshaw, Eds., ser. Lecture Notes in Computer Science, vol. 9215, Springer, 2015, pp. 161–185. DOI: 10.1007/978-3-662-47989-6_8. URL: https://doi.org/10.1007/978-3-662-47989-6%5C_8.

[142] Z. Eskandari, A. B. Kidmose, S. Kölbl, and T. Tiessen, "Finding integral distinguishers with ease," in *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, C. Cid and M. J. J. Jr., Eds., ser. Lecture Notes in Computer Science, vol. 11349, Springer, 2018, pp. 115–138. DOI: 10.1007/978-3-030-10970-7_6. URL: https://doi.org/10.1007/978-3-030-10970-7%5C_6.

[143] Z. Xiang, X. Zeng, and S. Zhang, "On the bit-based division property of s-boxes," *Science China Information Sciences*, vol. 65, no. 4, p. 149 101, May 2021, ISSN: 1869-1919. DOI: 10.1007/s11432-019-2634-7. URL: https://doi.org/10.1007/s11432-019-2634-7.

[144] X. Yu, W. Wu, Y. Li, and L. Zhang, "Cryptanalysis of reduced-round KLEIN block cipher," in *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, C. Wu, M. Yung, and D. Lin, Eds., ser. Lecture Notes in Computer Science, vol. 7537, Springer, 2011, pp. 237–250. DOI: 10.1007/978-3-642-34704-7_18. URL: https://doi.org/10.1007/978-3-642-34704-7%5C_18.

[145] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2021. URL: https://www.gurobi.com.

[146] S. Rasoolzadeh and H. Raddum, "Cryptanalysis of PRINCE with minimal data," in *Progress in Cryptology - AFRICACRYPT 2016 - 8th International Conference on Cryptology in Africa, Fes, Morocco, April 13-15, 2016, Proceedings*, D. Pointcheval, A. Nitaj, and T. Rachidi, Eds., ser. Lecture Notes in Computer Science, vol. 9646, Springer, 2016, pp. 109–126. DOI: 10.1007/978-3-319-31517-1_6. URL: https://doi.org/10.1007/978-3-319-31517-1%5C_6.

[147] F. Abed, E. List, and S. Lucks, "On the security of the core of PRINCE against biclique and differential cryptanalysis," *IACR Cryptol. ePrint Arch.*, p. 712, 2012. URL: http://eprint.iacr.org/2012/712.

[148] P. Morawiecki, "Practical attacks on the round-reduced PRINCE," *IET Inf. Secur.*, vol. 11, no. 3, pp. 146–151, 2017. DOI: 10.1049/iet-ifs.2015.0432. URL: https://doi.org/10.1049/iet-ifs.2015.0432.

[149] M. Eichlseder and D. Kales, "Clustering related-tweak characteristics: Application to MANTIS-6," *IACR Trans. Symmetric Cryptol.*, vol. 2018, no. 2, pp. 111–132, 2018. DOI: 10.13154/tosc.v2018.i2.111-132. URL: https://doi.org/10.13154/tosc.v2018.i2.111-132.

[150] S. Chen, R. Liu, T. Cui, and M. Wang, "Automatic search method for multiple differentials and its application on MANTIS," *Sci. China Inf. Sci.*, vol. 62, no. 3, 32111:1–32111:15, 2019. DOI: 10.1007/s11432-018-9658-0. URL: https://doi.org/10.1007/s11432-018-9658-0.

[151] T. Beyne, "Block cipher invariants as eigenvectors of correlation matrices," *J. Cryptol.*, vol. 33, no. 3, pp. 1156–1183, 2020. DOI: 10.1007/s00145-020-09344-1. URL: https://doi.org/10.1007/s00145-020-09344-1.

[152] I. Nikolic, L. Wang, and S. Wu, "The parallel-cut meet-in-the-middle attack," *Cryptogr. Commun.*, vol. 7, no. 3, pp. 331–345, 2015. DOI: 10.1007/s12095-014-0118-1. URL: https://doi.org/10.1007/s12095-014-0118-1.

[153] Z. Ahmadian, M. Salmasizadeh, and M. R. Aref, "Biclique cryptanalysis of the full-round KLEIN block cipher," *IET Inf. Secur.*, vol. 9, no. 5, pp. 294–301, 2015. DOI: 10.1049/iet-ifs.2014.0160. URL: https://doi.org/10.1049/iet-ifs.2014.0160.

[154] F. Abed, C. Forler, E. List, S. Lucks, and J. Wenzel, "Biclique cryptanalysis of the PRESENT and LED lightweight ciphers," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 591, 2012. URL: http://eprint.iacr.org/2012/591.

[155] J. Zhao, X. Wang, M. Wang, and X. Dong, "Differential analysis on block cipher PRIDE," *IACR Cryptol. ePrint Arch.*, vol. 2014, p. 525, 2014. URL: http://eprint.iacr.org/2014/525.

[156] Q. Yang, L. Hu, S. Sun, K. Qiao, L. Song, J. Shan, and X. Ma, "Improved differential analysis of block cipher PRIDE," in *Information Security Practice and Experience - 11th International Conference, ISPEC 2015, Beijing, China, May 5-8, 2015. Proceedings*, J. López and Y. Wu, Eds., ser. Lecture Notes in Computer Science, vol. 9065, Springer, 2015, pp. 209–219. DOI: 10.1007/978-3-319-17533-1_15. URL: https://doi.org/10.1007/978-3-319-17533-1%5C_15.

[157] I. Dinur, "Cryptanalytic time-memory-data tradeoffs for fx-constructions with applications to PRINCE and PRIDE," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, E. Oswald and M. Fischlin, Eds., ser. Lecture Notes in Computer Science, vol. 9056, Springer, 2015, pp. 231–253. DOI: 10.1007/978-3-662-46800-5_10. URL: https://doi.org/10.1007/978-3-662-46800-5%5C_10.

[158] Y. Dai and S. Chen, "Cryptanalysis of full PRIDE block cipher," *Sci. China Inf. Sci.*, vol. 60, no. 5, 052108:1–052108:12, 2017. DOI: 10.1007/s11432-015-5487-3. URL: https://doi.org/10.1007/s11432-015-5487-3.

[159] Q. Wang, Z. Liu, K. Varici, Y. Sasaki, V. Rijmen, and Y. Todo, "Cryptanalysis of reduced-round SIMON32 and SIMON48," in *Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings*, W. Meier and D. Mukhopadhyay, Eds., ser. Lecture Notes in Computer Science, vol. 8885, Springer, 2014, pp. 143–160. DOI: 10.1007/978-3-319-13039-2_9. URL: https://doi.org/10.1007/978-3-319-13039-2%5C_9.

[160] P. Mroczkowski and J. Szmidt, "The cube attack on stream cipher trivium and quadraticity tests," *Fundam. Informaticae*, vol. 114, no. 3-4, pp. 309–318, 2012. DOI: 10.3233/FI-2012-631. URL: https://doi.org/10.3233/FI-2012-631.

[161] P. Fouque and T. Vannet, "Improving key recovery to 784 and 799 rounds of trivium using optimized cube attacks," in *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, S. Moriai, Ed., ser. Lecture Notes in Computer Science, vol. 8424, Springer, 2013, pp. 502–517. DOI: 10.1007/978-3-662-43933-3_26. URL: https://doi.org/10.1007/978-3-662-43933-3%5C_26.

[162] C. Ye and T. Tian, "A new framework for finding nonlinear superpolies in cube attacks against trivium-like ciphers," in *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings*, W. Susilo and G. Yang, Eds., ser. Lecture Notes in Computer Science, vol. 10946, Springer, 2018, pp. 172–187. DOI: 10.1007/978-3-319-93638-3_11. URL: https://doi.org/10.1007/978-3-319-93638-3%5C_11.

[163] Y. Todo, T. Isobe, Y. Hao, and W. Meier, "Cube attacks on non-blackbox polynomials based on division property," in *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, J. Katz and H. Shacham, Eds., ser. Lecture Notes in Computer Science, vol. 10403, Springer, 2017, pp. 250–279. DOI: 10.1007/978-3-319-63697-9_9. URL: https://doi.org/10.1007/978-3-319-63697-9%5C_9.

[164] Q. Wang, Y. Hao, Y. Todo, C. Li, T. Isobe, and W. Meier, "Improved division property based cube attacks exploiting algebraic properties of superpoly," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, H. Shacham and A. Boldyreva, Eds., ser. Lecture Notes in Computer Science, vol. 10991, Springer, 2018, pp. 275–305. DOI: 10.1007/978-3-319-96884-1_10. URL: https://doi.org/10.1007/978-3-319-96884-1%5C_10.

[165] O. Kara and C. Manap, "A new class of weak keys for blowfish," in *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, A. Biryukov, Ed., ser. Lecture Notes in Computer Science, vol. 4593, Springer, 2007, pp. 167–180. DOI: 10.1007/978-3-540-74619-5_11. URL: https://doi.org/10.1007/978-3-540-74619-5%5C_11.

[166] P. Hawkes, "Differential-linear weak key classes of IDEA," in *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, K. Nyberg,

Ed., ser. Lecture Notes in Computer Science, vol. 1403, Springer, 1998, pp. 112–126. DOI: 10.1007/BFb0054121. URL: https://doi.org/10.1007/BFb0054121.

[167] F. Liu, T. Isobe, W. Meier, and K. Sakamoto, "Weak keys in reduced AEGIS and tiaoxin," *IACR Trans. Symmetric Cryptol.*, vol. 2021, no. 2, pp. 104–139, 2021. DOI: 10.46586/tosc.v2021.i2.104-139. URL: https://doi.org/10.46586/tosc.v2021.i2.104-139.

[168] R. Rohit and S. Sarkar, "Diving deep into the weak keys of round reduced ascon," *IACR Trans. Symmetric Cryptol.*, vol. 2021, no. 4, pp. 74–99, 2021. DOI: 10.46586/tosc.v2021.i4.74-99. URL: https://doi.org/10.46586/tosc.v2021.i4.74-99.

[169] G. Leander, M. A. Abdelraheem, H. AlKhzaimi, and E. Zenner, "A cryptanalysis of printcipher: The invariant subspace attack," in *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, P. Rogaway, Ed., ser. Lecture Notes in Computer Science, vol. 6841, Springer, 2011, pp. 206–221. DOI: 10.1007/978-3-642-22792-9_12. URL: https://doi.org/10.1007/978-3-642-22792-9%5C_12.

[170] G. Leander, B. Minaud, and S. Rønjom, "A generic approach to invariant subspace attacks: Cryptanalysis of robin, iscream and zorro," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, E. Oswald and M. Fischlin, Eds., ser. Lecture Notes in Computer Science, vol. 9056, Springer, 2015, pp. 254–283. DOI: 10.1007/978-3-662-46800-5_11. URL: https://doi.org/10.1007/978-3-662-46800-5%5C_11.

[171] Z. Li, X. Dong, and X. Wang, "Conditional cube attack on round-reduced ASCON," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 175–202, 2017. DOI: 10.13154/tosc.v2017.i1.175-202. URL: https://doi.org/10.13154/tosc.v2017.i1.175-202.

[172] Z. Li, W. Bi, X. Dong, and X. Wang, "Improved conditional cube attacks on keccak keyed modes with MILP method," in *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, T. Takagi and T. Peyrin, Eds., ser. Lecture Notes in Computer Science, vol. 10624, Springer, 2017, pp. 99–127. DOI: 10.1007/978-3-319-70694-8_4. URL: https://doi.org/10.1007/978-3-319-70694-8%5C_4.

[173] M. Lehmann and W. Meier, "Conditional differential cryptanalysis of grain-128a," in *Cryptology and Network Security, 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, J. Pieprzyk, A. Sadeghi, and M. Manulis, Eds., vol. 7712, Springer, 2012, pp. 1–11. DOI: 10.1007/978-3-642-35404-5_1. URL: https://doi.org/10.1007/978-3-642-35404-5%5C_1.

[174] Z. Ma, T. Tian, and W. Qi, "Improved conditional differential attacks on grain v1," *IET Inf. Secur.*, vol. 11, no. 1, pp. 46–53, 2017. DOI: 10.1049/iet-ifs.2015.0427. URL: https://doi.org/10.1049/iet-ifs.2015.0427.

[175] L. Karlsson, M. Hell, and P. Stankovski, "Not so greedy: Enhanced subset exploration for nonrandomness detectors," in *Information Systems Security and Privacy - Third International Conference, ICISSP 2017, Porto, Portugal, February 19-21, 2017, Revised Selected Papers*, P. Mori, S. Furnell, and O. Camp, Eds., ser. Communications in Computer and Information Science, vol. 867, Springer, 2017, pp. 273–294. DOI: 10.1007/978-3-319-93354-2_13. URL: https://doi.org/10.1007/978-3-319-93354-2%5C_13.

[176] J. Aumasson, I. Dinur, L. Henzen, W. Meier, and A. Shamir, "Efficient FPGA implementations of high-dimensional cube testers on the stream cipher grain-128," *IACR Cryptol. ePrint Arch.*, p. 218, 2009. URL: http://eprint.iacr.org/2009/218.

[177] S. Banik, S. Maitra, S. Sarkar, and M. S. Turan, "A chosen IV related key attack on grain-128a," in *Information Security and Privacy - 18th Australasian Conference, ACISP 2013, Brisbane, Australia, July 1-3, 2013. Proceedings*, C. Boyd and L. Simpson, Eds., ser. Lecture Notes in Computer Science, vol. 7959, Springer, 2013, pp. 13–26. DOI: 10.1007/978-3-642-39059-3_2. URL: https://doi.org/10.1007/978-3-642-39059-3%5C_2.

[178] Y. Todo, T. Isobe, W. Meier, K. Aoki, and B. Zhang, "Fast correlation attack revisited - cryptanalysis on full grain-128a, grain-128, and grain-v1," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, H. Shacham and A. Boldyreva, Eds., ser. Lecture Notes in Computer Science, vol. 10992, Springer, 2018, pp. 129–159. DOI: 10.1007/978-3-319-96881-0_5. URL: https://doi.org/10.1007/978-3-319-96881-0%5C_5.

[179] D. K. Dalai, S. Pal, and S. Sarkar, "Some conditional cube testers for grain-128a of reduced rounds," *IEEE Trans. Computers*, vol. 71, no. 6, pp. 1374–1385, 2022. DOI: 10.1109/TC.2021.3085144. URL: https://doi.org/10.1109/TC.2021.3085144.

[180] I. Dinur, T. Güneysu, C. Paar, A. Shamir, and R. Zimmermann, "An experimentally verified attack on full grain-128 using dedicated reconfigurable hardware," in *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, D. H. Lee and X. Wang, Eds., ser. Lecture Notes in Computer Science, vol. 7073, Springer, 2011, pp. 327–343. DOI: 10.1007/978-3-642-25385-0_18. URL: https://doi.org/10.1007/978-3-642-25385-0%5C_18.

[181] I. Dinur and A. Shamir, "Breaking grain-128 with dynamic cube attacks," in *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, A. Joux, Ed., ser. Lecture Notes in Computer Science, vol. 6733, Springer, 2011, pp. 167–187. DOI: 10.1007/978-3-642-21702-9_10. URL: https://doi.org/10.1007/978-3-642-21702-9%5C_10.