MILP-aided Cryptanalysis of Some Block Ciphers

Muhammad ElSheikh

A Thesis

in

The Concordia Institute

for

Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy (Information and Systems Engineering) at
Concordia University
Montréal, Québec, Canada

April 2021

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mr. Muhammad Hassan Gharieb Ahmed ElSheikh Ahmed**

Entitled: **MILP-aided Cryptanalysis of Some Block Ciphers**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

________________________________ Chair
Dr. Rajamohan Ganesan

________________________________ External Examiner
Dr. Huapeng Wu

________________________________ External to Program
Dr. Dongyu Qiu

________________________________ Examiner
Dr. Mohammad Mannan

________________________________ Examiner
Dr. Jeremy Clark

________________________________ Supervisor
Dr. Amr M. Youssef

Approved by ________________________________
Dr. Mohammad Mannan, Graduate Program Director

May 24th, 2021
Date of Defence

____
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

MILP-aided Cryptanalysis of Some Block Ciphers

Muhammad ElSheikh, Ph.D.

Concordia University, 2021

Symmetric-key cryptographic primitives, such as block ciphers, play a pivotal role in achieving confidentiality, integrity, and authentication – which are the core services of information security. Since symmetric-key primitives do not rely on well-defined hard mathematical problems, unlike public-key primitives, there are no formal mathematical proofs for the security of symmetric-key primitives. Consequently, their security is guaranteed only by measuring their immunity against a set of predefined cryptanalysis techniques, *e.g.,* differential, linear, impossible differential, and integral cryptanalysis.

The attacks based on cryptanalysis techniques usually include searching in an exponential space of patterns, and for a long time, cryptanalysts have performed this task manually. As a result, it has been hard, time-consuming, and an error-prone task. Indeed, the need for automatic tools becomes more pressing.

This thesis is dedicated to investigating the security of symmetric-key cryptographic primitives, precisely block ciphers. One of our main goals is to utilize Mixed Integer Linear Programming (MILP) to automate the evaluation and the validation of block cipher security against a wide range of cryptanalysis techniques. Our contributions can be summarized as follows.

First, we investigate the security of two recently proposed block ciphers, `CRAFT` and SPARX-128/256 against two variants of differential cryptanalysis. We utilize the simple

key schedule of CRAFT to construct several repeatable 2-round related-key differential characteristics with the maximum differential probability. Consequently, we are able to mount a practical key recovery attack on full-round CRAFT in the related-key setting. In addition, we use impossible differential cryptanalysis to assess SPARX-128/256 that is provable secure against single-trail differential and linear cryptanalysis. As a result, we can attack 24 rounds similar to the internal attack presented by the designers. However, our attack is better than the integral attack regarding the time and memory complexities.

Next, we tackle the limitation of the current Mixed Integer Linear Programming (MILP) model to automate the search for differential distinguishers through modular additions. The current model assumes that the inputs to the modular addition and the consecutive rounds are independent. However, we show that this assumption does not necessarily hold and the current model might lead to invalid attacks. Accordingly, we propose a more accurate MILP model that takes into account the dependency between consecutive modular additions. As a proof of the validity and efficiency of our model, we use it to analyze the security of Bel-T cipher—the standard of the Republic of Belarus.

Afterwards, we shift focus to another equally important cryptanalysis technique, *i.e.,* integral cryptanalysis using the bit-based division property (BDP). We present MILP models to automate the search for the BDP through modular additions with a constant and modular subtractions. Consequently, we assess the security of Bel-T block cipher against the integral attacks. Next, we analyze the security of the tweakable block cipher T-TWINE. We present key recovery attacks on 27 and 28 rounds of T-TWINE-80 and T-TWINE-128, respectively.

Finally, we address the limitation of the current MILP model for the propagation of the bit-based division property through large non-bit-permutation linear layers. The current models are either inaccurate, which might lead to missing some balanced bits, or inefficient in terms of the number of constraints. As a proof of the effectiveness of our approach, we improve the previous 3- and 4-round integral distinguishers of the Russian encryption standard—Kuznyechik, and the 4-round one of PHOTON's internal permutation (P_{288}). We also report a 4-round integral distinguisher for the Ukrainian standard Kalyna and a 5-round integral distinguisher for PHOTON's internal permutation (P_{288}).

Acknowledgments

Prophet Muhammad (SAWS) said: "*He who does not thank the people is not thankful to Allah*".

I would like to express my sincere appreciation to my supervisor Dr. Amr Youssef for his continuous motivation, guidance, patience and support during the Ph.D. journey.

Next, I would like to thank my lab mates for their friendship and support. Special thanks go to Ahmed Abdelkhalek and Mohamed Tolba for their constructive feedback and research discussions.

No words can express my gratitude and thanks to my mother and my father. Thank you my sister and my brother.

I wouldn't have made it without my lovely wife. Think you for your love, support, and encouragement during this journey.

Muhammad ElSheikh

To my little boys

Abdelrahman, Abdullah, and Alhassan

Table of Contents

List of Figures

List of Tables

List of Acronyms

AES	Advanced Encryption Standard
ARX	Add–Rotate–XOR
BDP	Bit-based Division Property
BFN	Balanced Feistel Network
CP	Constraint Programming
DDT	Differential Distribution Table
DES	Data Encryption Standard
GFS	Generalized Feistel Structure
MAC	Message Authentication Codes
MDS	Maximum Distance Separable
MILP	Mixed Integer Linear Programming
SAT	Boolean Satisfiability Problem
SMT	Satisfiability Modulo Theories
SPN	Substitution Permutation Network
TBC	Tweakable Block Cipher

Chapter 1

Introduction

1.1 Overview

Cryptology, the study of cryptosystems, could be imagined as a coin that has two faces, namely, cryptography and cryptanalysis. On the one hand, cryptography addresses the design issues of cryptosystems to achieve specific security goals. On the other hand, cryptanalysis guarantees that these goals have been established by trying to violate them. While the security goals considered in modern cryptography are application-specific, confidentiality, integrity, authentication, and non-repudiation form a framework upon which the others can be achieved [72]. The building blocks to construct cryptosystems are called cryptographic primitives.

As depicted in Figure 1.1, basic cryptographic primitives can be classified as unkeyed primitives, symmetric-key primitives, and public-key primitives.

Unkeyed primitives are aptly named; no key is used. Hash functions [76] are one of the most important commonly used cryptographic primitives that can be considered as unkeyed primitives. Using hash functions, we can compress a message of any arbitrary length into a hash value of fixed length. In other words, a hash function generates a fingerprint for the message where this fingerprint depends on all the bits of the message.

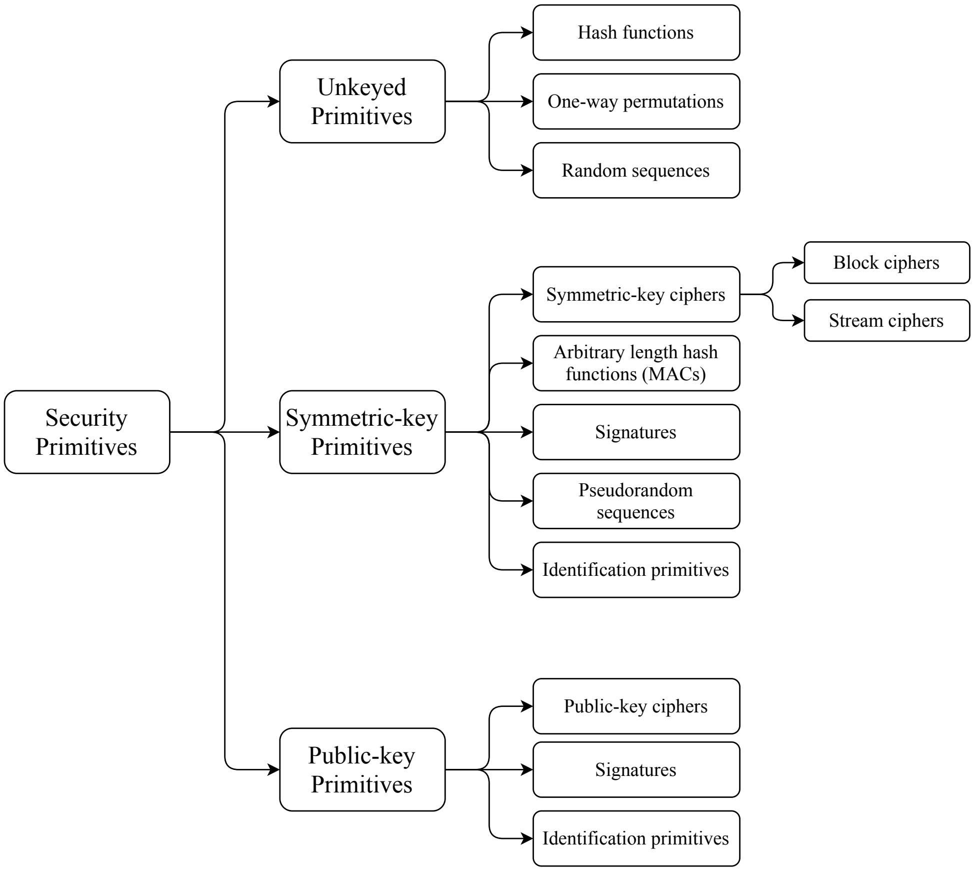

Figure 1.1: A taxonomy of cryptographic primitives [72]

Therefore, hash functions are heavily used to provide data integrity. While The cryptographic properties of hash functions could be application-specific, these properties should include preimage resistance, second preimage resistance, and collision resistance [76].

Symmetric-key primitives are cryptographic primitives where a secret key is shared between the communicating entities, *i.e.,* both sender and recipient use the same key. Block ciphers [58] are an example of these primitives. A block cipher consists of two algorithms namely, encryption and decryption. The encryption algorithm, used by a sender, takes two inputs; a block of data with a fixed length called plaintext and a secret key, and maps them into another block of data with the same length called ciphertext. The reverse process is performed using the decryption algorithm at the recipient, *i.e.,* the decryption algorithm maps the ciphertext back to its corresponding plaintext using the same shared secret key. Block ciphers can be used to provide data confidentiality.

Another example of symmetric-key primitives is Message Authentication Codes (MAC) [77] which can be used to provide both data integrity and authenticity. In MAC, a sender of a message computes an authentication tag of the message using a secret key, and then sends the tag along with the message. Then, the recipient re-computes the authentication tag corresponding to the received message using the same secret key, and then compares the re-computed tag against the received one. If the re-computed authentication tag matches the received one, then the sender is authenticated and the message integrity is verified. Unlike block ciphers, MAC primitives do not have to be invertible, as both sender and recipient perform the same operation to compute or verify the authentication tag.

Public-key primitives, also called asymmetric-key primitives, rely on a kind of trapdoor one-way functions. A trapdoor one-way function is a function that is easy to compute in one direction, but it is very difficult to invert without knowing special information called the trapdoor. In cryptosystems that use these primitives, each entity has two different keys, a public key and a private key. Since the private key is the trapdoor, it is kept secret. In contrast, the public key is known to all other entities without compromising security. A sender can encrypt a message using the recipient's public key. Therefore, the recipient is the only one who can decrypt the message because he/she knows the trapdoor of the function; the private key.

1.2 Motivation

Since public-key primitives are based on hard mathematical problems such as the discrete logarithm problem in the Diffie–Hellman [27] and the integer factorization problem in the RSA algorithm [79], their security is mathematically proven. In contrast, symmetric-key primitives do not rely on well-defined hard mathematical problems. Thus, there are no formal mathematical proofs for their security. Consequently, the security of symmetric-key cryptographic primitives is ensured only by measuring their immunity against a large set of predefined cryptanalysis techniques, *e.g.,* differential [12], linear [69], impossible differential [11], zero-correction [19], and integral cryptanalysis [57]. Among all symmetric-key primitives, block ciphers are the most commonly used ones. This is because block ciphers can be also used as building blocks for other cryptographic primitives such as hash functions and MAC schemes. Thus, evaluating the security of block ciphers is a major aspect of the design process of secure block ciphers.

Furthermore, performing cryptanalysis techniques consists of two steps: finding a distinguisher that covers some rounds to distinguish this reduced-round block cipher from a random permutation, then converting this distinguishing attack to a key recovery attack. Finding that distinguisher is usually the hardest and time-consuming step. This step usually includes searching in an exponential space of patterns, and for a long time, cryptanalysts have performed this task manually. As a result, it becomes more and more hard, time-consuming, and error-prone. With the increasing complexity in modern symmetric-key ciphers, the need for automated tools becomes more pressing. Such tools can play a significant role, not only in cryptanalysis but also in the design of these symmetric key-primitives.

1.3 Thesis Contributions and Outline

In this thesis, we investigate the security of several block ciphers that are either recently proposed or standardized by a standardization body, *e.g.,* `CRAFT` [7], SPARX-128/256 [28], Bel-T [10], T-TWINE [81], Kuznyechik [86], and Kalyna [74]. Moreover, with the help of Mixed Integer Linear Programming (MILP), we are able to automate the search process for finding distinguishers in differential cryptanalysis and integral attacks based on the bit-based division property.

After providing a formal definition of block ciphers and a brief overview about some cryptanalysis techniques in Chapter 2, we focus in the first part of this thesis (Chapters 3, 4, and 5) on differential cryptanalysis and some of its variants. Afterwards, we shift our focus in the second part (Chapters 6, 7, and 8) to developing and applying MILP-aided integral attacks based on the bit-based division property. The contributions of this thesis are as follows:

- Chapter 3 presents our security assessment of the lightweight block cipher `CRAFT`. Precisely, we utilize the simple key schedule of `CRAFT` to propose a systematic method for constructing several repeatable 2-round related-key differential characteristics with the maximum differential probability 2^{-2}. We then employ these characteristics to mount a practical key recovery attack on full-round `CRAFT` in the related-key setting.

- Chapter 4 provides our impossible differential attack against SPARX-128/256 block cipher. Firstly, we present two 20-round impossible differential distinguishers. Then, we use them to launch a key recovery attack against 24 rounds. Although our attack covers the same number of rounds that the designers reach using the integral cryptanalysis, our attack achieves better time and memory complexities.

- Chapters 5 and 6 provide our security assessment of Bel-T-256 which is a member of the Bel-T block cipher family that has been adopted recently as the national standard of the Republic of Belarus. In Chapter 5, we firstly propose our new MILP model to describe the differential propagation through the modular addition/subtraction taking into account the dependency between the consecutive modular operations. Then, the proposed MILP model is used to find a 3-round distinguisher. Finally, we employ this distinguisher to launch a key recovery attack against Bel-T-256. In Chapter 6, we present MILP models to automate the search for the bit-based division property through the modular addition with a constant and the modular subtractions. Consequently, we assess the security of Bel-T-256 against integral attacks.

- Chapter 7 presents integral attacks on the two variants of the tweakable block cipher T-TWINE. In particular, we study the implications of adding the tweak to the structure of TWINE [94]. Then, we present several 19-round integral distinguishers in both chosen tweak-plaintext and chosen tweak-ciphertext settings. Accordingly, we mount key recovery attacks against 27 and 28 rounds of T-TWINE-80 and T-TWINE-128, respectively.

- Chapter 8 introduces our new MILP model for the propagation of the bit-based division property through large non-bit-permutation linear layers. The models in the literature are either inaccurate or inefficient. As a proof of the effectiveness of our model, we improve the previous 3- and 4-round integral distinguishers of the Russian encryption standard — Kuznyechik, and the 4-round one of PHOTON's internal permutation (P_{288}) [47]. We also report a 4-round integral distinguisher for the Ukrainian standard Kalyna and a 5-round integral distinguisher for PHOTON's internal permutation (P_{288}).

The above contributions have been published in [31, 33–35, 37, 38]. Other works conducted during the tenure of this Ph.D. have been published in [3, 32, 36, 42, 100].

Chapter 2

Background

In this chapter, we provide a brief overview of block ciphers focusing on some of their relevant cryptanalytic techniques.

2.1 Block Ciphers

One of the most widely used symmetric-key primitives is block ciphers which can be considered as keyed permutations. More precisely, a block cipher is a deterministic algorithm used to map a plaintext P to a ciphertext C, and vice versa, under a specific secret key K. The formal definition of the block cipher is as follows:

Definition 2.1 (Block cipher[[72], Definition 7.1]) *An n-bit block cipher is a function $E : V_n \times \mathcal{K} \rightarrow V_n$, such that for each key $K \in \mathcal{K}, E(P, K)$ is an invertible mapping (the encryption function for K) from V_n to V_n, written $E_K(P)$. The inverse mapping is the decryption function, denoted $D_K(C)$. $C = E_K(P)$ denotes the ciphertext C resulting from encrypting plaintext P under K.*

The concept of iterated round function is dominant to many modern block ciphers where a simple weak key-dependent round function is iterated several times in order to achieve a higher degree of security in terms of confusion and diffusion. Figure 2.1 gives a schematic view of an iterative block cipher. The formal definition of an iterative block cipher is as follows:

Definition 2.2 (Iterative block cipher[[17], Definition 4]) *An n-bit block cipher E is called an iterative block cipher with r rounds if for each key it can be represented as a composition of keyed round permutations, that is, if for each* $K \in \mathbb{F}_2^k$*:*

$$E(.,K) = f_{r-1}(.,K_{r-1}) \circ f_{r-2}(.,K_{r-2}) \circ \cdots \circ f_1(.,K_1) \circ f_0(.,K_0),$$

where $\circ$ *denotes the superposition of permutations,* $f_i(.,K_i) : \mathbb{F}_2^n \to \mathbb{F}_2^n$ *are key-dependent round permutations,* K_i *are round subkeys derived from the secret key* K *using a key schedule algorithm:*

$$k_s : K \to (K_0, K_1, \cdots, K_{r-2}, K_{r-1}).$$

The iterated block cipher is an iterative block cipher with an identical round permutation, *i.e.,* $f_i = f$. Typically, there are two widely used structures of the iterated round function, namely Balanced Feistel Network (BFN) (*e.g.,* DES [26]) and Substitution Permutation Network (SPN) (*e.g.,* AES [30]) as depicted in Figure 2.2.

2.2 Block Ciphers Security

According to the definition of perfect security introduced by Shannon [85], the key used in a perfectly secure block cipher must have entropy higher than or at least equal to the entropy of the plaintext. In other words, the length of the key must be at least equal to the length of the plaintext and it cannot be reused to encrypt another data (one-time use) [25]. With the large amounts of data needed to be encrypted, such perfectly secure block ciphers become impractical. Instead, we can use the concept of computational security to achieve practical block ciphers. Computational security is defined as follows:

Definition 2.3 (Computational security [[25], Definition 7]) *A block cipher E us-*

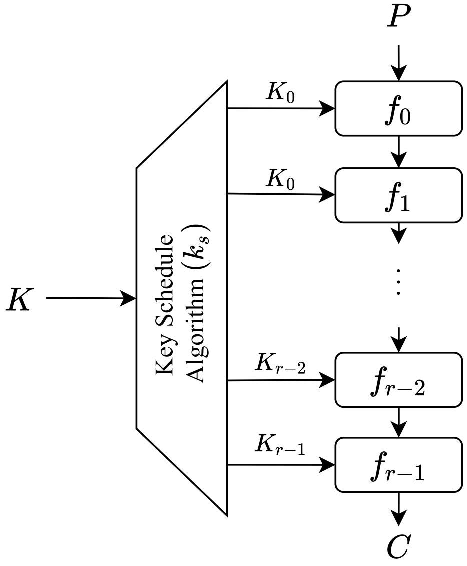

Figure 2.1: An iterative block cipher

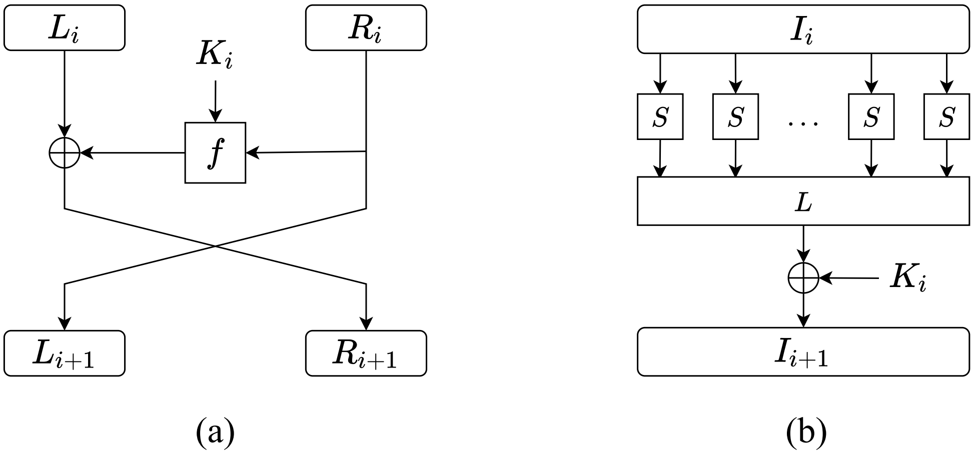

Figure 2.2: Round function of (a) Balanced Feistel Network, and (b) Substitution-Permutation Network

ing a k-bit secret key is called computationally secure if there exist no attacks on E with a complexity less than the one of an exhaustive key search, i.e., 2^k, where the complexity of the attack comprises the time (work factor), memory (storage requirement), and data (type and amount of data) complexities required to perform the attack.

According to the above definition, it is difficult to prove that a block cipher is computationally secure. Instead, we typically evaluate the security of block ciphers by considering their immunity against a large set of predefined cryptanalytic attacks. During this evaluation, we use the following criteria, based on the definition of computational security, to consider the effectiveness of different attacks:

1. Data complexity: the number of plaintexts and/or ciphertexts required for launching the attack.

2. Memory (storage) complexity: the amount of memory used during the attack.

3. Time (computational) complexity: the amount of computation or time required for executing the attack.

2.2.1 Attack Models

Attack models reflect the kind of available information to an attacker and which operations could be applied by the attacker on the plaintexts and/or ciphertexts. Attack models can be categorized as:

- Ciphertext-only: A set of ciphertexts is available to the attacker without knowing the corresponding plaintexts.

- Known-plaintext: A set of both plaintexts and their corresponding ciphertexts is available to the attacker.

- Chosen-plaintext (ciphertext): A set of plaintexts (ciphertexts) chosen by the attacker and their corresponding ciphertexts (plaintexts) is available to the attacker.

- Adaptive chosen-plaintext: The attacker can adapt the next chosen-plaintext based on some intermediate results obtained from the previous chosen-plaintext and its corresponding ciphertext. This adaptation is performed during the attack.

- Adaptive chosen-ciphertext: Similar to the previous model, but the attacker can adapt the next chosen-ciphertext.

- Related-key: Combined with chosen-plaintext (ciphertext) model, the attacker is able to choose the relation between the unknown master secrete key and the key used by the oracle to encrypt (decrypt) the chosen-plaintext (ciphertext).

2.3 Cryptanalytic Techniques

In this section, we discuss some of the most powerful attacks to evaluate the security of block ciphers.

2.3.1 Differential Cryptanalysis

Differential cryptanalysis, which was introduced by Biham and Shamir with applications on DES in 1990 [12], is one of the most powerful attacks that are used to evaluate the security of symmetric-key primitives. Differential cryptanalysis is a chosen-plaintext attack in which an attacker takes the advantage of existing a pair of inputs with a specific difference that is highly correlated to their output difference after some rounds. In other words, for an n-bit primitive, the attacker looks for a pair of inputs with input difference ΔX_0 that gives, after r rounds, another pair with output difference ΔX_r with a probability (p) higher than 2^{-n}, independent of the secret key. The pair of differences $(\Delta X_0, \Delta X_r)$ associated with the probability (p) is called an r-round differential.

Using this differential, a key recovery attack can be performed by appending (prepending) some rounds after (before) the differential and guessing the round keys. Therefore, finding this differential is essential for the attack. Since this task is not easy for many ciphers, the attacker instead searches for a differential path with a significantly high probability through linear and nonlinear components of the cipher under attack. In other words, the attacker tries to construct a good differential trail:

$$\Delta X_0 \xrightarrow{p_0} \Delta X_1 \xrightarrow{p_1} \cdots \xrightarrow{p_{i-1}} \Delta X_i \xrightarrow{p_i} \cdots \xrightarrow{p_{r-1}} \Delta X_r$$

where ΔX_i is the difference at the intermediate state of the cipher after i-round, and p_i is the probability of this differential trail in round i. Consequently, the probability of this trail can be approximately calculated as the multiplication of its intermediate probabilities ($\prod_{i=0}^{r-1} p_i$), assuming the independence between them. As a result, the r-round differential ($\Delta X_0 \xrightarrow{p} \Delta X_r$) can be considered as the set of all possible differential trails that have the same input and output differences, and the probability of this differential can be calculated as the sum of the probabilities of all these individual differential trails. However, the enumeration of all possible differential trails is almost infeasible. Therefore, obtaining one or two trails with a significantly high probability will be enough to lunch the attack in many cases.

Since the propagation of the difference through the linear elements is deterministic, the round function probability (p_i) relies propagating the difference through the nonlinear elements and how many of them are active (has nonzero input/output difference). When the number of active nonlinear elements is low, the differential probability will be high and then the data complexity will be low.

There are many variants of the basic form of differential cryptanalysis such as truncated differential cryptanalysis [55], higher-order differential cryptanalysis [61], boomerang cryptanalysis [102], and impossible differential cryptanalysis [11] that uses a differential characteristic of probability zero to exclude wrong keys.

2.3.2 Linear Cryptanalysis

Linear cryptanalysis is another example of the powerful attacks that are used to assess symmetric-key primitives. Matsui and Yamagishi applied this technique for the first time to evaluate the security of FEAL [70] and then Matsui was able to break the full round of DES using this technique [69]. Linear cryptanalysis is a known-plaintext attack and its main idea relies on finding a linear approximation between some of the input and output bits over a round-reduced version of a block cipher by linearizing the nonlinear elements. Due to the nonlinear component, this approximation will hold with probability p and if this probability is significantly high or low, *i.e.,* the probability closes to 1 or 0, then the block cipher can be distinguished from a random permutation in which this approximation holds with probability 0.5. More formally, for an r-round cipher with n-bit block size, suppose (X_0, X_r) denotes the input at round 0 and the output at round r, the linear approximation can be expressed as:

$$\Gamma_0 \odot X_0 \oplus \Gamma_r \odot X_r = 0$$

where $\odot$ is the scalar product over $\mathbb{F}_2$, and Γ_i is an n-bit vector called a linear mask, whose elements belong to $\mathbb{F}_2$. The linear mask Γ_i is used to specify which bits in X_i are involved in the linear approximation.

A linear approximation with probability (p), written as $(\Gamma_0 \xrightarrow{\epsilon} \Gamma_r)$ and so-called an r-round linear hull, is characterized by its bias $\epsilon = |p - \frac{1}{2}|$ which measures how much the cipher is diverted from a random permutation, which has zero bias. Similar to the r-round differential, the most efficient method to construct a good linear approximation is by searching for a chain of linear masks $(\Gamma_0, \Gamma_1, \ldots, \Gamma_r)$, called a linear trail, through the nonlinear component of cipher's rounds such that the sub-linear approximation $(\Gamma_i \odot X_i \oplus \Gamma_{i+1} \odot X_{i+1} = 0)$ holds with bias $\epsilon_i > 0$. Accordingly, the overall bias of the linear trail can be computed using the piling-up lemma [69] as $(2^{r-1} \prod_{i=0}^{r-1} \epsilon_i)$ under the same assumptions mentioned above in differential cryptanalysis. As a result, the r-round linear hull $(\Gamma_0 \xrightarrow{\epsilon} \Gamma_r)$ can be considered as a set of all possible linear trails that have the same input and output linear masks, and the overall squared bias ϵ^2 can be calculated as the sum of the squared bias of each individual linear trail in this set.

The distinguishing attack described above can be turned into a key recovery attack, like in differential cryptanalysis, by appending some rounds before or after the distinguisher and guessing the round keys. The data complexity of the linear cryptanalysis is inversely proportional to the squared bias ϵ^2. Therefore, the minimum the number of active nonlinear elements, the maximum the bias, and the minimum the data complexity.

Similar to differential cryptanalysis, many extensions and improvements of linear cryptanalysis have been proposed. For example, the use of chosen-plaintexts instead of known-plaintexts was proposed to reduce the data complexity [59]. In the same manner, the multidimensional linear attacks were introduced in [49] to reduce the data complexity. Analogous to impossible differential cryptanalysis, the concept of zero-correlation cryptanalysis is proposed in [19] where a linear hull with zero bias, *i.e.,* its probability is exactly 0.5 is exploited in order to exclude wrong keys.

2.3.3 Impossible Differential Cryptanalysis

Impossible differential cryptanalysis is a variant of differential cryptanalysis as mentioned earlier. It was introduced independently by Biham *et al.* [11] and Knudsen [56]. While differential cryptanalysis exploits differentials with the highest probability, impossible differential cryptanalysis relies on a differential with a probability exactly equal to zero *i.e.,* an attacker looks for a differential with an input difference that can never lead to any particular output difference.

Miss-in-the-Middle is a general technique to construct an impossible differential distinguisher where a block cipher E is divided into two cascaded sub-ciphers E_0 and E_1 such that $E = E_1 \circ E_0$, as depicted in Figure 2.3. Each sub-cipher has a differential with a probability equal to 1. These two differentials are concatenated to construct an impossible differential distinguisher covers E as follows. Suppose there exists a differential $(\alpha \xrightarrow{p=1} \beta)$ through E_0, and there exists another differential $(\delta \xrightarrow{p=1} \gamma)$ through $(E_1)^{-1}$. When the intermediate differences (β, γ) do not mach, the input difference (α) can not lead to the difference δ at the output of E and we have an impossible differential $(\alpha \not\to \delta)$

Once an attacker finds this impossible differential, a key recovery attack can be launched by prepending and/or appending a few additional rounds (called analysis rounds)

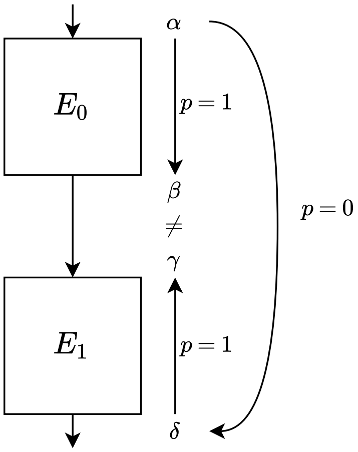

Figure 2.3: Visualization of Miss-in-the-Middle technique.

before and/or after the distinguisher. The attack proceeds in two phases as follows. In the first phase, the attacker collects a set of plaintext/ciphertext pairs with certain differences. In the second phase, the attacker guesses some key bits involved in the analysis rounds and checks whether the guessed key leads any pair of the collected ones to the impossible differential. If this happened, the guessed key bits must be wrong. Accordingly, the attacker can discard as many wrong keys as possible and exhaustively search on the remaining key bits. One method to improve the time complexity of the attack is by using the early abort technique [67] that allows us to guess some of the involved key bits, not all of them, and discard unuseful pairs as early as possible and therefore reducing the time complexity of the attack.

2.3.4 Integral Cryptanalysis

Daemen *et al.* proposed a new cryptanalysis technique to analyze the security of the block cipher SQUARE [23]. Subsequently, Knudsen and Wagner [57] formalized this technique and called it *integral attack*. The integral attack has several variants with different names such as collision attack [45], multiset attack [15], and saturation attack [68].

The integral attack starts by finding an integral distinguisher where a set of plaintexts is chosen such that it has a constant value at some bits while the other bits vary through all possible values. This set gives after r rounds another set. If the XOR sum of all bits (or some of them) on the new set is always 0 irrespective of the used secret key, we denote these bits as balanced bits and conclude that the block cipher under test has an r-round integral distinguisher.

The techniques to construct an integral distinguisher include estimating the algebraic degree of the nonlinear parts of the cipher, and evaluating the propagation of the following integral properties [57]:

- ALL ($\mathcal{A}$) : Every member appears the same number of times in the set.
- BALANCE ($\mathcal{B}$) : The XOR of all members in the set is 0.
- CONSTANT ($\mathcal{C}$) : The value is fixed to a constant for all members in the set.
- UNKNOWN ($\mathcal{U}$) : The set is indistinguishable from one of n-bit random values.

The propagation rules of these integral properties can be summarized as the following:

Though Linear layer

$$\begin{cases} \mathcal{A} \oplus \mathcal{A} \rightarrow \mathcal{B} \\ \mathcal{A} \oplus \mathcal{B} \rightarrow \mathcal{B} \\ \mathcal{A} \oplus \mathcal{U} \rightarrow \mathcal{U} \\ \mathcal{B} \oplus \mathcal{U} \rightarrow \mathcal{U} \end{cases}$$

Though non-linear Function($\mathcal{F}$)

$$\begin{cases} \mathcal{F}(\mathcal{A}) \rightarrow \mathcal{A} \\ \mathcal{F}(\mathcal{B}) \rightarrow \mathcal{U} \\ \mathcal{F}(\mathcal{U}) \rightarrow \mathcal{U} \end{cases}$$

A key recovery attack can be launched based on the integral distinguisher as follows. First, we append some additional rounds after the distinguisher. Then, we collect the set of plaintexts and obtain the corresponding ciphertexts. After that, for each set of plaintexts/ciphertexts, we guess some key bits involved in the analysis rounds and partially decrypt the ciphertexts to reach the distinguisher. If the set of the partially decrypted ciphertexts does not satisfy the balance condition *i.e.,* have XOR sum of 1, then the guessed key bits must be wrong. Like in impossible differential cryptanalysis, we can discard as many wrong keys as possible and exhaustively search over the remaining key bits.

2.3.5 Division Property

The division property is a generalized integral property that utilizes the hidden properties between the traditional $\mathcal{A}$ and $\mathcal{B}$ by exploiting the algebraic degree of the nonlinear components of block ciphers [95]. Since it was proposed by Todo at Eurocrypt 2015, it has become one of the most efficient methods to build integral distinguishers. It has been used to analyze the security claims of many symmetric-key primitives, *e.g.,* the full round MISTY1 was broken using a 6-round integral distinguisher found by the division property [96]. The division property was succeeded by a more precise version called the *bit-based division property* (BDP) in [98] which exploits the internal structure of the nonlinear components to analyze block ciphers at the bit level.

Definition 2.4 (Bit product function [95]) *For two n-bit vectors $\boldsymbol{x}$ and $\boldsymbol{u}$, the bit product function $\pi_{\boldsymbol{u}}(\boldsymbol{x}) : \mathbb{F}_2^n \to \mathbb{F}_2$ is defined as*

$$\pi_{\boldsymbol{u}}(\boldsymbol{x}) = \prod_{i=0}^{n-1} x_i^{u_i}$$

where x_i and u_i are the i-th bits of $\boldsymbol{x}$ and $\boldsymbol{u}$, respectively.

Definition 2.5 (Bit-based Division Property [98]) *Let $\mathbb{X}$ be a multiset whose elements take a value of $\mathbb{F}_2^n$. When the multiset $\mathbb{X}$ has the division property $\mathcal{D}_{\mathbb{K}}^n$, where $\mathbb{K}$ denotes a set of n-dimensional vectors whose i-th element takes 0 or 1, it fulfills the* *following conditions for any* $\boldsymbol{u} \in \mathbb{F}_2^n$*:*

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

Even though the BDP is more accurate and can find better integral distinguishers, handling its propagation is computationally intensive. The first search tool utilized the bit-based division property was limited to building integral distinguishers for block ciphers with block size less than 32 bits since the complexity of the search is around $\mathcal{O}(2^n)$ where n is the block size [98]. Xiang *et al.* [104] have overcome the problem of the restriction on the block size by proposing the *division trails.* Using the division trial, the search process for an integral distinguisher can be converted to checking whether a specific division trail exists or not.

Definition 2.6 (Division Trail [104]) *Let f denote the round function of an iterated block cipher. Assume that the input multiset to the block cipher has the initial division property $\mathcal{D}^n_{\{\boldsymbol{k}\}}$, and denote the division property after i-round propagation through f by $\mathcal{D}^n_{\mathbb{K}_i}$. Thus, we have the following chain of division property propagations: $\{\boldsymbol{k}\} \overset{\text{def}}{=} \mathbb{K}_0 \xrightarrow{f} \mathbb{K}_1 \xrightarrow{f} \mathbb{K}_2 \xrightarrow{f} \cdots \xrightarrow{f} \mathbb{K}_r$. Moreover, for any vector $\boldsymbol{k}_i^* \in \mathbb{K}_i (i \geq 1)$, there must exist a vector $\boldsymbol{k}_{i-1}^* \in \mathbb{K}_{i-1}$ such that $\boldsymbol{k}_{i-1}^*$ can propagate to $\boldsymbol{k}_i^*$ by the division property propagation rules. Furthermore, for $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r) \in \mathbb{K}_0 \times \mathbb{K}_1 \times \cdots \times \mathbb{K}_r$, if $\boldsymbol{k}_{i-1}$ can propagate to $\boldsymbol{k}_i$ for all $i \in \{1, 2, \ldots, r\}$, we call $(\boldsymbol{k}_0, \boldsymbol{k}_1, \ldots, \boldsymbol{k}_r)$ an r-round division trail.*

Using the division trial, the search process for an integral distinguisher is converted to check if the division trail $\boldsymbol{k_0} \rightarrow \cdots \rightarrow e_i$ (a unit vector whose i-th element is 1) does exist or not. If it does not exist, then the i-th bit of r-round output is balanced.

In the following, we summarize the propagation rules of the BDP through the basic operations. Suppose $\mathbb{K}_{\boldsymbol{x}}$ and $\mathbb{K}_{\boldsymbol{y}}$ denote the input and output division property of the function $\boldsymbol{y} = f(\boldsymbol{x})$.

- **Rule of COPY**: let $\boldsymbol{x} = (x_0)$ and $\boldsymbol{y} = (y_0, y_1)$ where $(y_0, y_1) = (x_0, x_0)$. Therefore,

$$\begin{cases} \mathbb{K}_{\boldsymbol{y}} = \{(0,0)\}, & if\ \mathbb{K}_{\boldsymbol{x}} = \{(0)\} \\ \mathbb{K}_{\boldsymbol{y}} = \{(1,0),(0,1)\}, & if\ \mathbb{K}_{\boldsymbol{x}} = \{(1)\} \end{cases}$$

- **Rule of XOR**: let $\boldsymbol{x} = (x_0, x_1)$ and $\boldsymbol{y} = (y_0)$ where $y_0 = x_0 \oplus x_1$. Therefore,

$$\begin{cases} \mathbb{K}_{\boldsymbol{y}} = \{(0)\}, & if\ \mathbb{K}_{\boldsymbol{x}} = \{(0,0)\} \\ \mathbb{K}_{\boldsymbol{y}} = \{(1)\}, & if\ \mathbb{K}_{\boldsymbol{x}} = \{(0,1)\} \\ \mathbb{K}_{\boldsymbol{y}} = \{(1)\}, & if\ \mathbb{K}_{\boldsymbol{x}} = \{(1,0)\} \end{cases}$$

- **Rule of AND**: let $\boldsymbol{x} = (x_0, x_1)$ and $\boldsymbol{y} = (y_0)$ where $y_0 = x_0 x_1$. Therefore,

$$\begin{cases} \mathbb{K}_{\boldsymbol{y}} = \{(0)\}, & if\ \mathbb{K}_{\boldsymbol{x}} = \{(0,0)\} \\ \mathbb{K}_{\boldsymbol{y}} = \{(1)\}, & if\ \mathbb{K}_{\boldsymbol{x}} = \{(0,1)\} \\ \mathbb{K}_{\boldsymbol{y}} = \{(1)\}, & if\ \mathbb{K}_{\boldsymbol{x}} = \{(1,0)\} \\ \mathbb{K}_{\boldsymbol{y}} = \{(1)\}, & if\ \mathbb{K}_{\boldsymbol{x}} = \{(1,1)\} \end{cases}$$

2.3.6 MILP-aided Cryptanalysis

As mentioned earlier, attacks based on cryptographic techniques usually include searching for a distinguisher that is used to differentiate the output of a block cipher from the output of a random permutation. In general, the process of finding this distinguisher includes searching for a specific pattern in exponential space where the meaning of this pattern is different from one technique to another. In general, the core idea behind many automated tools is to convert this search problem into a well-defined optimization/satisfiability problem, then utilize some existing solvers to catch a solution. The automated tools can be divided into the following three categories:

- Boolean Satisfiability Problem (SAT)/Satisfiability Modulo Theories(SMT) [54, 89].
- Constraint Programming (CP) [43, 44].
- Mixed Integer Linear Programming (MILP) [22, 73, 82, 88, 92, 104].

Mixed Integer Linear Programming (MILP). Mixed Integer Linear Programming (MILP) problem is a mathematical optimization problem whose variables are restricted to be integers and its goal is to optimize a linear objective function with respect to a set of linear constraints. For example:

$$maximize\ (x + y + z), \text{ subject to} \begin{cases} x + y + 5z \leq 2 \\ x + y \geq 1 \\ x, y, z\ as\ binary. \end{cases}$$

MILP-aided Differential Cryptanalysis. The first attempt to utilize MILP technique in symmetric-key cryptanalysis was developed by Mouha *et al.* [73] in which they applied a MILP technique to prove security bounds against both differential and linear cryptanalysis. Later, Sun *et al.* [91] proposed MILP models that represent exactly the propagation of the differential through SPN block ciphers in order to automate the search for high probability (related-key) differential or linear characteristics. Cui *et al.* [22] proposed a MILP model for both impossible differential and zero-correlation attacks. Sasaki and Todo [82] developed a new search tool for impossible differential using MILP. One of the downsides of these MILP models was the inability to efficiently describe the Difference Distribution Table (DDT) of large (8-bit) S-boxes which was tackled by Abdelkhalek *et al.* [1]. Regarding ARX-based block ciphers, Fu *et al.* [41] represented the conditions developed by Lipmaa and Moriai [65] by a set of MILP constraints in order to automate the search for the best differential trail through the modular addition. Throughout the work presented in this thesis, we study the model by Fu *et al.* in details and highlight some limitations on it. Then, we propose a more accurate model for the propagation of the XOR difference through modular addition and subtraction.

MILP-aided Bit-based Division Property. With the help of the division trail, it becomes easy to employ MILP for constructing the integral distinguisher using the BDP. Xiang *et al.* [104] provided an accurate model for the propagation of the BDP through the basic operations; `COPY`, `XOR`, and `AND`, in addition to an accurate model for S-boxes. Sun *et al.* complement this work by handling ARX-based ciphers (modular addition operation) [88]. Recently, Todo *et al.* utilize MILP model of the BDP to improve the cube attacks [97]. For ciphers with non-bit-permutation linear layers, Sun *et al.* [90] proposed a model relying on decomposing the matrix corresponding to the linear layer into its basic operations; `COPY` and `XOR`. Another model for the propagation of the BDP through non-bit-permutation linear layers is presented by Zhang and Rijmen in [106]. Throughout the work presented in this thesis, we complement the work by Sun *et al.* [88] for ARX-based ciphers by proposing MILP models for the propagation of the BDP through a modular addition with a constant and a modular subtraction. Moreover, we propose a new MILP model for the propagation of the BDP through large non-bit-permutation linear layers.

Chapter 3

Related-key Differential Cryptanalysis of Full Round CRAFT

CRAFT [7] is a lightweight tweakable block cipher introduced at FSE 2019. In this chapter, we utilize the simple key schedule of CRAFT to propose a systematic method for constructing several repeatable 2-round related-key differential characteristics with probability 2^{-2}. We then employ one of these characteristics to mount a key recovery attack on full-round CRAFT using 2^{31} queries to the encryption oracle, 2^{85} encryptions, and 2^{41} 64-bit blocks of memory. Additionally, we manage to use 8 related-key differential distinguishers, with 8 related-key differences, in order to mount a key recovery attack on the full-round cipher with $2^{35.17}$ queries to the encryption oracle, 2^{32} encryptions and about 2^{6} 64-bit blocks of memory. Furthermore, we present another attack that recovers the whole master key with $2^{36.09}$ queries to the encryption oracle and only 11 encryptions with 2^{7} blocks of memory using 16 related-key differential distinguishers.

3.1 Introduction

Modern symmetric-key cryptographic primitives such as the Advanced Encryption Standard (AES), which are likely designed for desktops and servers, cannot be easily implemented on resource-constrained devices, *e.g.,* sensor networks, healthcare equipment, Internet of Things (IoT) devices, and RFIDs. With the rapidly increasing demand for such devices, the National Institute for Standards and Technology (NIST) has initiated a standardization process for new lightweight cryptographic algorithms for use in resource-constrained devices. SKINNY [6], PRESENT [18], SIMON [5], and GIFT [4] are examples of such lightweight block ciphers that have been recently proposed.

At FSE 2019, Beierle *et al.* presented CRAFT [7], a new lightweight tweakable block cipher with a block size of 64 bits and a key length of 128 bits associated with 64 bits as a tweak. One of the main design criteria of CRAFT is the efficient protection of its implementations against differential fault analysis. In the design paper, the authors provide the security analysis of CRAFT against several cryptanalysis techniques such as differential, linear, impossible differential, zero correlation, and integral cryptanalysis in the single-key and related-tweak settings. They also presented a deterministic related-key/related-tweak differential characteristic. However, this characteristic cannot be used to mount a key recovery attack. In this chapter, we study in details the security of CRAFT against the related-key differential attack. More precisely,

1. We utilize the simple key schedule of CRAFT to present a systematic method of how to select the key difference in addition to the input and the output differences of the 2-round structure of CRAFT such that the input difference is the same as the output difference. Thus, the resulting 2-round characteristic is repeatable. In the same time, we also try to maximize the probability of that characteristic. Thereby, we use it as a building block for constructing a longer characteristic. To illustrate the effectiveness of this method, we present 17 repeatable 2-round characteristics, each one of them has only one active S-box and holds with probability equals to the maximum differential probability of an active S-box of CRAFT (2^{-2}).

2. We extend one of these characteristics to a 28-round related-key differential characteristic with probability 2^{-28}. After that, we employ it to mount a key recovery

attack on full-round CRAFT using 2^{31} queries to the encryption oracle and 2^{85} encryptions, and 2^{41} 64-bit blocks of memory.

3. We can speed up the key recovery attack against the full-round CRAFT using $2^{35.17}$ queries to the encryption oracle and 2^{32} full-round encryptions. To this end, we manage to use 8 different related-key differential characteristics (with 8 related-key differences) in order to recover 96 bits from the secret master key and then we get the full master key by testing the right 96-bit key along with the remaining 32 bits of the key using a plaintext/ciphertext pair.

4. Furthermore, we can perform the previous attack without the exhaustive search step and recover the whole master key with $2^{36.09}$ queries to the encryption oracle and only 11 full-round encryptions (instead of 2^{32} in the above attack) using 16 different related-key differential characteristics (with 16 related-key differences). This attack has been verified experimentally.

The rest of this chapter is organized as follows. In Section 3.2, we briefly revisit the specifications of CRAFT. A systematic method to build a repeatable 2-round related-key characteristic is explained in Section 3.3. In Section 3.4, we describe the key recovery attack against the full rounds of CRAFT using a single related-key differential characteristic. Then, the details of our attack using multiple related-key differential characteristics are presented in Section 3.5. Finally, the chapter is summarized in Section 3.6.

3.2 Specifications of CRAFT

CRAFT [7] is a lightweight tweakable block cipher with a block size of 64 bits, a key length (K) of 128 bits, and a tweak (T) of 64 bits. The internal state of the cipher can be represented as a 4×4 square array of nibbles or as a 16-nibble vector by concatenating the rows of the square array. The notation $I_{i,j}$ is used to denote the nibble located at row

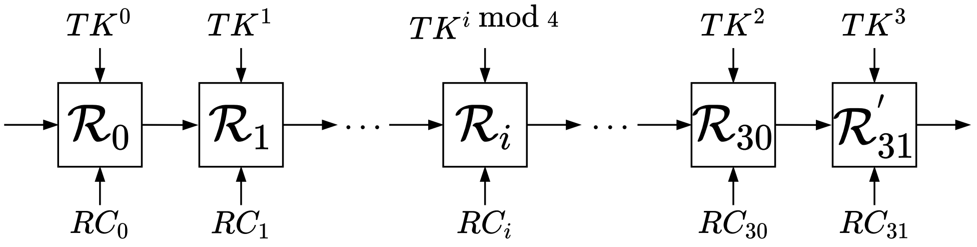

Figure 3.1: Structure of `CRAFT`

i and column j of the 4×4 array. Also, a single subscript I_i denotes the nibble in the i-th position of 16-nibble vector, *i.e.*, $I_{i,j} = I_{4i+j}$.

Tweakey Schedule. The 128-bit key K is split into two 64-bit subkeys K^0 and K^1. Similar to the internal state, the subkeys K^0 and K^1 in addition to the 64-bit input tweak T are represented as 4×4 square arrays of nibbles or as a 16-nibble vectors using a similar indexing technique as for the internal state. Then, four 64-bit tweakeys TK^0, TK^1, TK^2 and TK^3 are derived from K^0 and K^1 with the associated T as follows:

$$TK^0 = K^0 \oplus T, \quad TK^1 = K^1 \oplus T, \quad TK^2 = K^0 \oplus Q(T), \quad TK^3 = K^1 \oplus Q(T).$$

where $Q(T)$ is a permutation on the nibbles of the input tweak T using a permutation $\mathcal{Q} = [12, 10, 15, 5, 14, 8, 9, 2, 11, 3, 7, 4, 6, 0, 1, 13]$. In other words, the i-th nibble of $Q(T)$ ($T(Q)_i$, $0 \leq i \leq 15$) is equal to the $\mathcal{Q}(i)$-th nibble of T ($Q(T)_i = T_{\mathcal{Q}(i)}$). The tweakey $TK^{i \bmod 4}$ ($0 \leq i \leq 31$) is used during the i-th round of the encryption operation in order to update the internal state.

Encryption Operation. The encryption operation proceeds as follows. First, the plaintext $m = m_0||m_1|| \cdots ||m_{14}||m_{15}$ (where m_i is a 4-bit nibble) is loaded into the internal state. Then, the internal state is updated by applying the full round function of `CRAFT` 31 times ($\mathcal{R}_i$, $0 \leq i \leq 30$). Finally, one more linear round($\mathcal{R}'_{31}$) is applied on the internal state to compute the ciphertext as shown in Figure 3.1, where RC_i is the round constant.

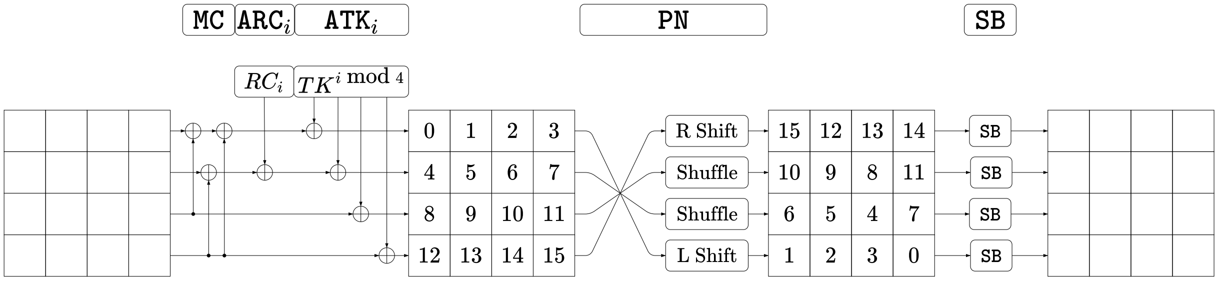

Figure 3.2: One full round function of CRAFT

The full round of CRAFT ($\mathcal{R}_i$) consists of the following five operations: MixColumn, AddConstant$_i$, AddTweakey$_i$ PermuteNibbles and SubBox as described in Figure 3.2. The last round ($\mathcal{R}'_{31}$) omits PermuteNibbles and SubBox operations from the full round. The used five operations are defined as follows:

- MixColumn (MC): Each column of the internal state is multiplied by a binary matrix M where

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

  This operation can be described using the XOR operation as follows. For each column j $(0 \leq j \leq 3)$,

$$\begin{bmatrix} I_{0,j} \\ I_{1,j} \\ I_{2,j} \\ I_{3,j} \end{bmatrix} \mapsto \begin{bmatrix} I_{0,j} \oplus I_{2,j} \oplus I_{3,j} \\ I_{1,j} \oplus I_{3,j} \\ I_{2,j} \\ I_{3,j} \end{bmatrix}$$

- AddConstants$_i$ (ARC$_i$): In the i-th round $(0 \leq i \leq 31)$, the internal state nibbles I_4 and I_5 are XOR-ed with the two nibbles (a and b), respectively, where a and b represented the 2-nibble round constant $RC_i = (a, b)$. These round constants are generated using 4-bit and 3-bit LFSRs. The details of generating the round constants can be found in [7].

Table 3.1: 4-bit S-box of CRAFT

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

- $\texttt{AddTweakey}_i$ ($\texttt{ATK}_i$): Each nibble of the internal state is XOR-ed with the corresponding nibble of the tweakey $TK^{i \bmod 4}$.
- PermuteNibbles (PN): An permutation $\mathcal{P}$ is applied on the nibble positions of the internal state. In particular, for all $0 \leq i \leq 15$, I_i is replaced by $I_{\mathcal{P}(i)}$, where

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

- SubBox (SB): A nonlinear bijective mapping applied on every nibble of the internal state in parallel using the S-box given in Table 3.1.

3.3 Related-key Differential Characteristic of CRAFT

In this section, we describe our technique to build a repeatable 2-round related-key characteristic with a high probability p. A repeatable characteristic is a characteristic where the input difference is the same as the output difference. Hence, we can construct a long characteristic by repeating the short one n times and the probability of the long one will be p^n.

Denote the state at the input and the output of round i of CRAFT by x^i and x^{i+1}, respectively, and the state after MC, $\texttt{ARC}_i$ and $\texttt{ATK}_i$ operations by y^i. Thus we have

$$y^i = \texttt{ATK}_i \circ \texttt{ARC}_i \circ \texttt{MC}(x^i)$$
$$x^{i+1} = \texttt{SB} \circ \texttt{PN}(y^i)$$

In the related-key with a single tweak setting, the tweak (T) has zero difference and the subkeys (K^0, K^1) have the nonzero differences ΔK^0 and ΔK^1, respectively. Thereby, the four tweaks have nonzero differences as follows:

$$\Delta TK^0 = \Delta TK^2 = \Delta K^0, \qquad \Delta TK^1 = \Delta TK^3 = \Delta K^1$$

A 2-round Characteristic. Consider two consecutive rounds of $\texttt{CRAFT}$, i and $i+1$, where i is even. Thus $\Delta TK^{i \bmod 4} = \Delta K^0$ and $\Delta TK^{(i+1) \bmod 4} = \Delta K^1$. We start building a repeatable 2-round characteristic by setting the input and the output differences (Δx^i and Δx^{i+2}) of the 2-round with arbitrary nonzero values such that $\Delta x^i = \Delta x^{i+2}$. Then, we deterministically propagate the input difference Δx^i forward through the $\texttt{MC}$ and $\texttt{ARC}_i$ operations and choose ΔK^0 such that $\Delta K^0 = \texttt{ARC}_i \circ \texttt{MC}(\Delta x^i)$. Thereby, we ensure that $\Delta y^i = 0$, $\Delta x^{i+1} = 0$ and $\Delta y^{i+1} = \Delta K^1$. From the other direction, we propagate the output difference Δx^{i+2} backward through $\texttt{SB}$ and $\texttt{PN}$ operations to obtain Δy^{i+1} and select ΔK^1 such that $\Delta K^1 = \Delta y^{i+1} = \texttt{PN}_i^{-1} \circ \texttt{SB}^{-1}(\Delta x^{i+2})$. It should be noted that the probability of propagating Δx^{i+2} backward to ΔK^1 is the same as the probability of propagating ΔK^1 forward to Δx^{i+2} due to the properties of the S-box of $\texttt{CRAFT}$. Therefore, the overall probability of this characteristic depends on the probability of propagating Δx^{i+2} through $\texttt{SB}^{-1}$ operation. In order to maximize the overall probability, we have to minimize the number of active nibbles in the input/output differences to only one active nibble with a difference value (α). Therefore, ΔK^1 also has a single active nibble with a difference value (β) such that $\Pr[\texttt{SB}^{-1}(\alpha) \rightarrow \beta] = p$. Finally, we select the value of the tuple (α, β) so that p is equal to the maximum differential probability for an active S-box which is 2^{-2}.

Figure 3.3 depicts an example of such characteristics in which we set the input/output differences to zero except for the two nibbles Δx^i_{12} and Δx^{i+2}_{12}, which we set to α. Therefore, we select the difference of the subkey K^0 such that it has zero difference except the nibbles ΔK^0_0, ΔK^0_4 and ΔK^0_{12} have a nonzero difference (α). For the subkey K^1, it will have zero difference in 15 nibbles and nonzero difference β in the nibble

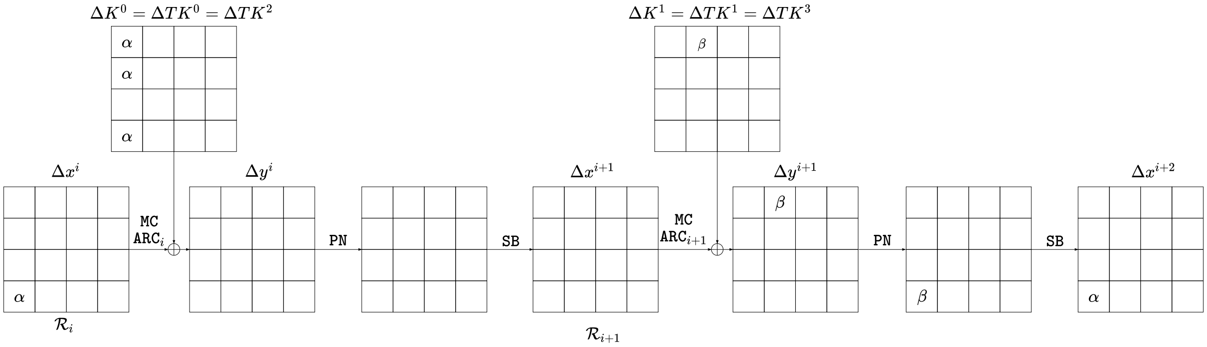

Figure 3.3: A repeatable 2-round related-key characteristic of `CRAFT` with probability 2^{-2}.

ΔK^1_1 such that $\Pr[\texttt{SB}^{-1}(\alpha) \rightarrow \beta] = 2^{-2}$.

Based on the differential distribution table (DDT) of the `CRAFT`'s S-box, the unordered tuples (α, β) can take one of the values from the following set:

$$\begin{aligned}(\alpha,\beta) \text{ or } (\beta,\alpha) \in \{&(\texttt{1},\texttt{2}),(\texttt{2},\texttt{4}),(\texttt{2},\texttt{9}),(\texttt{2},\texttt{c}),(\texttt{3},\texttt{6}),(\texttt{5},\texttt{7}),(\texttt{5},\texttt{a}),\\ &(\texttt{7},\texttt{d}),(\texttt{a},\texttt{a}),(\texttt{a},\texttt{d}),(\texttt{a},\texttt{f}),(\texttt{b},\texttt{b}),(\texttt{e},\texttt{e}),(\texttt{f},\texttt{f})\}.\end{aligned} \tag{3.1}$$

We can also build a repeatable 2-round characteristic by setting the input and the output differences to zero differences ($\Delta x^i = \Delta x^{i+2} = 0$), then selecting ΔK^0 such that it has only one active nibble with nonzero difference (α). After that, we obtain the value of the difference ΔK^1 which has only one active nibble with nonzero difference (β) such that $\Delta K^1 = \texttt{ARC}_{i+1} \circ \texttt{MC} \circ \texttt{SB} \circ \texttt{PN}(\Delta K^0)$. Finally, we select the value of the tuple (α, β) from the previously mentioned set. Table 3.2 summarizes some examples for the obtained 2-round related-key differential characteristics.

In the following sections, we utilize the repeatable 2-round related-key differential characteristics derived here to mount two key recovery attacks against the full round of `CRAFT`.

Table 3.2: Examples of repeatable 2-round related-key differential characteristics of CRAFT, all of them hold with probability 2^{-2} starting from an even round i. and (α, β) can take one of the values given by equation (3.1).

	$\Delta K^0 = \Delta TK^0 = \Delta TK^2$	$\Delta K^1 = \Delta TK^1 = \Delta TK^3$	$\Delta x^i = \Delta x^{i+2}$
$\mathbf{RK}_0$	$(0,0,0,\alpha,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,\beta,0,0,0,\beta,0,0,0,0,0,0,0,\beta,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\mathbf{RK}_1$	$(\alpha,0,0,0,\alpha,0,0,0,0,0,0,0,\alpha,0,0,0)$	$(0,\beta,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,\alpha,0,0,0)$
$\mathbf{RK}_2$	$(0,\alpha,0,0,0,\alpha,0,0,0,0,0,0,0,\alpha,0,0)$	$(0,0,\beta,0,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,\alpha,0,0)$
$\mathbf{RK}_3$	$(0,0,\alpha,0,0,0,\alpha,0,0,0,0,0,0,0,\alpha,0)$	$(0,0,0,\beta,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,\alpha,0)$
$\mathbf{RK}_4$	$(0,0,0,\alpha,0,0,0,\alpha,0,0,0,0,0,0,0,\alpha)$	$(\beta,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,\alpha)$
$\mathbf{RK}_5$	$(\alpha,0,0,0,0,0,0,0,\alpha,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,\beta,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,\alpha,0,0,0,0,0,0,0)$
$\mathbf{RK}_6$	$(0,\alpha,0,0,0,0,0,0,0,\alpha,0,0,0,0,0,0)$	$(0,0,0,0,0,\beta,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,\alpha,0,0,0,0,0,0)$
$\mathbf{RK}_7$	$(0,0,\alpha,0,0,0,0,0,0,0,\alpha,0,0,0,0,0)$	$(0,0,0,0,\beta,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,\alpha,0,0,0,0,0)$
$\mathbf{RK}_8$	$(0,0,0,\alpha,0,0,0,0,0,0,0,\alpha,0,0,0,0)$	$(0,0,0,0,0,0,0,\beta,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,\alpha,0,0,0,0)$
$\mathbf{RK}_9$	$(0,0,0,0,\alpha,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,\beta,0,0,0,0,0,0)$	$(0,0,0,0,\alpha,0,0,0,0,0,0,0,0,0,0,0)$
$\mathbf{RK}_{10}$	$(0,0,0,0,0,\alpha,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,\beta,0,0,0,0,0,0)$	$(0,0,0,0,0,\alpha,0,0,0,0,0,0,0,0,0,0)$
$\mathbf{RK}_{11}$	$(0,0,0,0,0,0,\alpha,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,\beta,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,\alpha,0,0,0,0,0,0,0,0,0)$
$\mathbf{RK}_{12}$	$(0,0,0,0,0,0,0,\alpha,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,\beta,0,0,0,0)$	$(0,0,0,0,0,0,0,\alpha,0,0,0,0,0,0,0,0)$
$\mathbf{RK}_{13}$	$(\alpha,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,\beta)$	$(\alpha,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\mathbf{RK}_{14}$	$(0,\alpha,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,\beta,0,0,0)$	$(0,\alpha,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\mathbf{RK}_{15}$	$(0,0,\alpha,0,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,\beta,0,0)$	$(0,0,\alpha,0,0,0,0,0,0,0,0,0,0,0,0,0)$
$\mathbf{RK}_{16}$	$(0,0,0,\alpha,0,0,0,0,0,0,0,0,0,0,0,0)$	$(0,0,0,0,0,0,0,0,0,0,0,0,0,0,\beta,0)$	$(0,0,0,\alpha,0,0,0,0,0,0,0,0,0,0,0,0)$

3.4 Related-key Differential Attack Using Single Difference

In this section, we employ the repeatable 2-round characteristic ($\mathbf{RK}_0$) with the tuple $(\alpha, \beta) = (\mathtt{4}, \mathtt{2})$ to present a related-key differential attack against the full round of `CRAFT`. By repeating $\mathbf{RK}_0$ (14) times as depicted in Figure 3.4, we are able to construct a 28-round related-key differential characteristic (covered from round 0 to round 27) with probability $(2^{-2})^{14} = 2^{-28}$. We have verified this characteristic experimentally.

Since the characteristic ends at x^{28} where all nibbles have zero differences, we propagate this difference through the last 4 rounds and obtain the difference at the ciphertext (ΔC) in the form of

$$(\delta_4, \delta_3, \delta_9, \delta_6, \delta_4, 0, \delta_8, \delta_6, 0, \delta_3, 0, 0, \delta_4, 0, \delta_7, \delta_6).$$

Thus, we can derive the following conditions on the ciphertext (ΔC):

$$\begin{aligned}
&\Delta C_5 = \Delta C_8 = \Delta C_{10} = \Delta C_{11} = \Delta C_{13} = \mathtt{0},\\
&\Delta C_1 = \Delta C_9,\\
&\Delta C_0 = \Delta C_4 = \Delta C_{12},\\
&\Delta C_3 = \Delta C_7 = \Delta C_{15}.
\end{aligned}$$

Our attack has two phases: Data Collection phase and Key Recovery phase.

3.4.1 Data Collection

We select a set of 2^m 64-bit plaintexts associated with a 64-bit tweak in which the plaintexts and the tweak can take any arbitrary values. Each plaintext is encrypted twice, using the secret master key ($K^0||K^1$) and using the secret master key XORed with the key differences $((K^0 \oplus \Delta K^0)||(K^1 \oplus \Delta K^1))$. Then, we compute the difference at

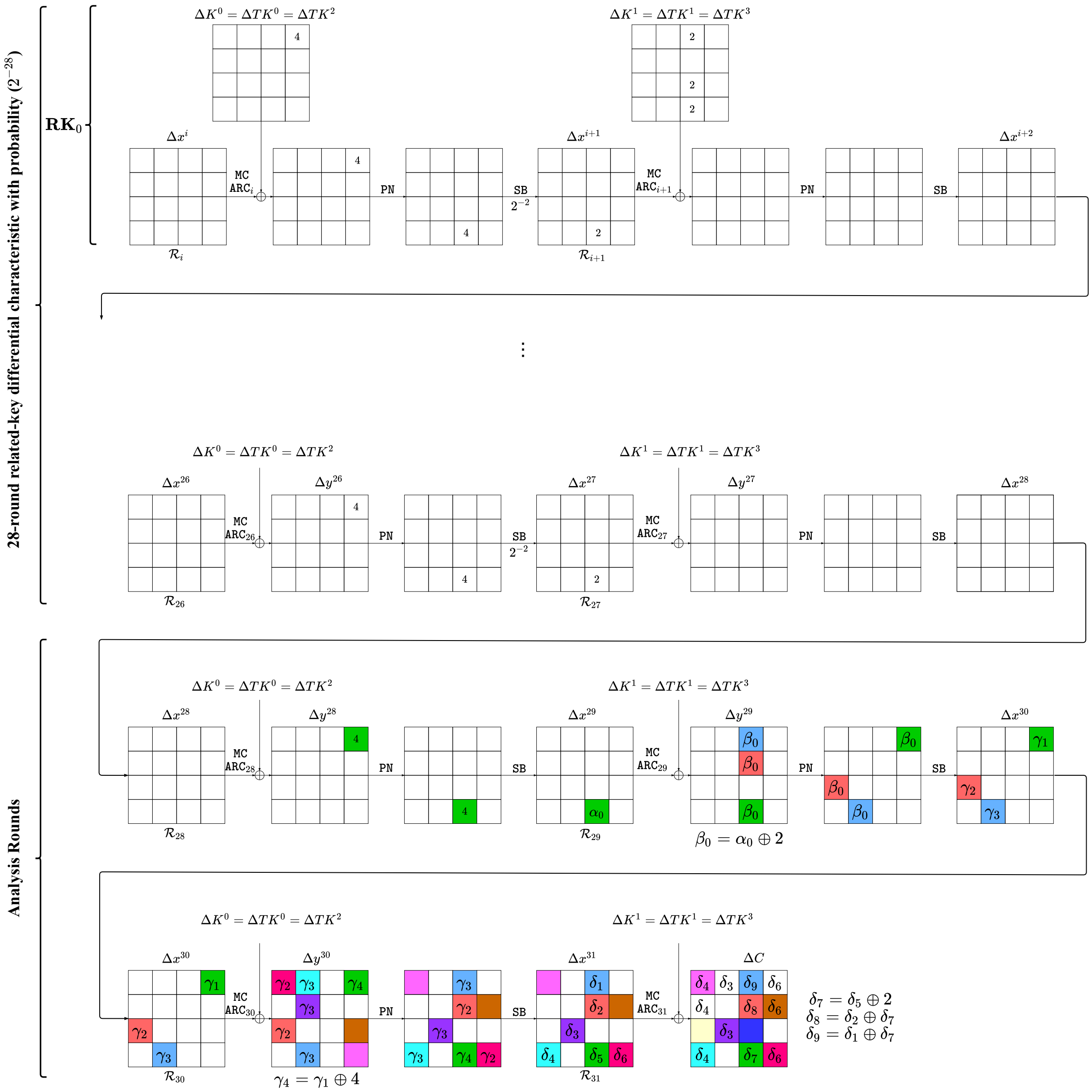

Figure 3.4: The related-key differential attack against Full `CRAFT` using the repeatable 2-round related-key differential characteristic ($\mathbf{RK}_0$) where the colored cells are known values and differences.

the ciphertext (ΔC) and filter out the plaintext/ciphertext pairs that do not satisfy the conditions, obtained above, on ΔC. This step provides a $5 \times 4 + 4 + 2 \times 4 + 2 \times 4 = 40$ bits filtration. Suppose the number of the remaining plaintext/ciphertext pairs after this filtration is $2^{m'}$, then on average, $2^{m'} = 2^m \times 2^{-40} = 2^{m-40}$.

3.4.2 Key Recovery

We first prepare $2^{11\times 4} = 2^{44}$ counters corresponding to the 44 bits of the key involved in the analysis. After that, for each ciphertext pair in the filtered $2^{m'}$ pairs obtained in the data collection phase, we apply the following procedure:

1. Guess the key nibbles (K_9^1, K_{12}^1) and partially decrypt the ciphertext to obtain the differences $(\Delta y_1^{30}, \Delta y_5^{30})$. The average number of the guessed keys that satisfy the condition $(\Delta y_1^{30} = \Delta y_5^{30})$ is $2^{2\times 4} \times 2^{-4} = 2^4$.

2. Guess the key nibbles $(K_6^1, K_{14}^1, K_{15}^1)$ and partially decrypt the ciphertext to obtain the values and differences at the nibbles $(y_0^{30}, y_3^{30}, y_8^{30})$ and discard any key that does not lead to satisfy the condition of $(\Delta y_0^{30} = \Delta y_8^{30})$. The average number of the keys passing this filtration is $2^4 \times 2^{3\times 4} \times 2^{-4} = 2^{12}$.

3. Guess the value of $(K_2^1 \oplus K_{10}^1)$ with associated value of K_{14}^1 passed the filtration on the previous step (step 2) and partially decrypt the ciphertext to obtain the value and the difference at the nibble (y_{13}^{30}). Then filter out the keys if the difference(Δy_{13}^{30}) is not the same as the differences $(\Delta y_1^{30}, \Delta y_5^{30})$ that are obtained in the step (1). Thus, the average number of keys suggested by a pair after this step is $2^{12} \times 2^4 \times 2^{-4} = 2^{12}$.

4. Guess the key nibbles (K_8^0, K_{13}^0) and partially decrypt the nibbles (y_8^{30}, y_{13}^{30}) obtained on steps (2,3), respectively, and get the differences $(\Delta y_2^{29}, \Delta y_6^{29})$. The average number of the guessed keys that satisfy the condition of $(\Delta y_2^{29} = \Delta y_6^{29})$ is $2^{12} \times 2^{2\times 4} \times 2^{-4} = 2^{16}$.

5. Guess the key nibble (K_7^1) and use the previous guessed value of K_{15}^1 to partially decrypt the ciphertext in order to obtain the value of y_{11}^{30}. Also, guess the value of $(K_0^1 \oplus K_8^1)$ and use the previous guess of K_{12}^1 to obtain the value of y_{15}^{30}. The average number of keys suggested by a pair after this step is $2^{16} \times 2^{2\times 4} = 2^{24}$.

6. Use the value and the difference at (y_3^{30}) from step (2) with the values (y_{11}^{30}, y_{15}^{30}) obtained from the previous step to get the value and the difference at (y_{14}^{29}) by guessing the value of ($K_3^0 \oplus K_{11}^0 \oplus K_{15}^0$). We then filter out the keys if the difference(Δy_{14}^{29}) is not the same as the differences ($\Delta y_2^{29}, \Delta y_6^{29}$) that are obtained in the step (4). Thus, the average number of keys suggested by a pair after this step is $2^{24} \times 2^4 \times 2^{-4} = 2^{24}$.

7. Use the previously guessed value of the key nibble (K_{14}^1) to partially decrypt the nibble y_{14}^{29} to obtain the difference Δy_3^{28} and discard the keys if the condition of ($\Delta y_3^{28} = 4$) is not satisfied. Consequently, the average number of keys suggested by a pair after this procedure will be decreased to $2^{24} \times 2^{-4} = 2^{20}$. Thus, we increment the corresponding 2^{20} counters.

After repeating the above procedure for $2^{m'}$ pairs, we select the key corresponding to the highest counter as a 44-bit right key. Then, we recover the 128-bit master key by testing the 44-bit right key along with the remaining 84 bits of the master key that are not involved in the analysis using 2 plaintext/ciphertext pairs.

3.4.3 Attack Complexity and Success Probability

In what follows, we present the complexity analysis of the attack in order to determine the required number of chosen plaintexts and the memory required to launch this attack.

Data Complexity. We utilize the concept of signal-to-noise ratio (S/N) [13] in order to determine the required number of chosen plaintext/ciphertext pairs (2^m). $S/N = \frac{2^k \times p}{\omega \times \phi}$, where k is the number of key bits involved in the analysis, p is the probability of the differential characteristic, ω is the number of guessed keys by a pair, and ϕ is the ratio of the pairs that are not discarded. In our analysis, $k = 44$, $p = 2^{-28}$, $\omega = 2^{20}$, and $\phi = 2^{-40}$. Therefore, we have $S/N = \frac{2^{44} \times 2^{-28}}{2^{20} \times 2^{-40}} = 2^{36}$. Due to this high S/N, we can use the recommendation of Biham and Shamir [13] that $3 \sim 4$ right pairs are sufficient enough to mount a successful differential attack. Therefore, we select the number of plaintext/ciphertext pairs (2^m) equal to $4 \times p^{-1} = 2^{30}$. Consequently, the data complexity will be 2^{31} chosen plaintexts.

During the data collection phase, we discard the pairs that do not satisfy the conditions on the differences of the ciphertext. The probability of satisfying these conditions is 2^{-40}, *i.e.,* there are, on average, $2^{m-40} = 2^{30-40} = 2^{-10}$ remaining pairs. This means that the right pairs only pass this filtration and $2^{m'} = 4$.

According to [84] and due to the high S/N, the success probability of the attack (P_s) can be calculated as $P_s \approx \Phi(\sqrt{p \times 2^m})$ where Φ is the cumulative distribution function of the standard normal distribution. Therefore, our differential attack will succeed with probability $P_s \approx 0.9772$.

Time Complexity. During the key recovery phase, we perform several partial decryption of some nibbles in which we can consider each nibble decryption as $\frac{1}{16}$ of 1-round decryption. The dominant time complexity of the key recovery procedure comes from step 6 in which we perform $2^{m'} \times 2^4 \times 2^{24} \times 2 = 2^{31}$ partial decryption of one nibble. This time equals to $\frac{1}{16} \times \frac{1}{32} \times 2^{31} = 2^{22}$ 32-round encryptions. Then, we perform the exhaustive search over the remaining 2^{84} keys using 2 plaintext/ciphertext pairs. The time complexity of this step is $2 \times 2^{84} = 2^{85}$ 32-round encryptions. Therefore, the total time complexity of the attack is $2^{22} + 2^{85} \approx 2^{85}$ encryptions.

Memory Complexity. The dominant part of the memory complexity comes from storing 2^{44} counters. Since the upper limit of each counter is $2^{m'} = 4$, we can store each counter in one byte. Therefore, we need $2^{44} \times \frac{8}{64} = 2^{41}$ 64-bit blocks of memory.

3.5 Related-key Differential Attack Using Multiple Differences

In this section, we present a key recovery attack in the related-key model against the full-round CRAFT with $2^{35.17}$ queries to the encryption oracle and 2^{32} full-round encryptions. To this end, we manage to use 8 different related-key differential characteristics in order to recover 96 bits (represented in 24 nibbles) from the secret master key and then we get the full master key by testing the right 96-bit key along with the remaining 32 bits of the key using 2 plaintext/ciphertext pairs. Moreover, we can omit the exhaustive search step and recover the whole master key with $2^{36.09}$ queries to the encryption oracle and only 11 full-round encryptions.

30-round Related-key Differential Characteristics. We employ the repeatable 2-round characteristics ($\mathbf{RK}_1 - \mathbf{RK}_8$) (see Table 3.2) with the tuple $(\alpha, \beta) = (\mathtt{4}, \mathtt{2})$ in order to build eight 30-round characteristics as follows. First, we repeat each $\mathbf{RK}_i$ $(1 \leq i \leq 8)$ 14 times to build a 28-round characteristic with probability $(2^{-2})^{14} = 2^{-28}$. Then, we append another 2 rounds with probability of (2^{-2}). Thus, we are able to construct a 30-round characteristic with total probability (p) of 2^{-30}. Figure 3.5 depicts the 30-round characteristic that is built using $\mathbf{RK}_1$.

Consequently, we use these characteristics one by one to collect 8 datasets $(\mathcal{D}_i, 1 \leq i \leq 8)$ (Data Collection phase) and then apply a partial-key recovery process to determine a part of the master secret key (Key Recovery phase).

3.5.1 Data Collection

We use the 30-round characteristic based on the repeatable 2-round characteristic, *e.g.,* $\mathbf{RK}_1$ to build the dataset $\mathcal{D}_1$ as follows. This characteristic ends at x^{30} with zero differences in all nibbles except $\Delta x^{30}_{12} = \mathtt{1}$ as depicted in Figure 3.5. After that, by propagating this difference through the last two rounds, we are able to obtain the difference at the ciphertext (ΔC) in the form

$$(0, \delta_0, \beta_0, \gamma_0, 0, 0, 0, \gamma_0, 0, 0, \beta_0, 0, 0, 0, 0, \gamma_0)$$

where $\delta_0 = \alpha_0 \oplus 2$ and based on the DDT of `CRAFT`'s S-box, $\alpha_0, \beta_0, \gamma_0 \in \{\mathtt{0}, \mathtt{4}, \mathtt{7}, \mathtt{9}, \mathtt{a}, \mathtt{c}\}$. Thus, we can derive the following conditions on the difference of the ciphertext:

$$\Delta C_i = \mathtt{0},\ i \in \{0, 4, 5, 6, 8, 9, 11, 12, 13, 14\}, \qquad \Delta C_1 = \delta_0,$$

$$\Delta C_2 = \Delta C_{10} = \beta_0, \qquad \Delta C_3 = \Delta C_7 = \Delta C_{15} = \gamma_0.$$

Consequently, we first select a set of $4 \times p^{-1} = 4 \times 2^{30} = 2^{32}$ arbitrary plaintexts ($\mathcal{L}_0$) and then we create another set of 2^{32} plaintexts ($\mathcal{L}_1$) by XORing each plaintext in the first set $\mathcal{L}_0$ with the input difference. After encrypting the two sets ($\mathcal{L}_0, \mathcal{L}_1$) using $(K^0||K^1)$ and $((K^0 \oplus \Delta K^0)||(K^1 \oplus \Delta K^1))$, respectively, we discard the pairs when the output difference does not match the required output difference (ΔC). The probability of getting (ΔC) is $2^{-(10\times4+4+2\times4)} \times (\frac{6}{16})^3 \approx 2^{-56.25}$. In other words, only the right pairs can pass this filtration. Thus, we collect, on average, 4 right pairs that follow the characteristic.

We repeat the same approach using the same set of plaintexts ($\mathcal{L}_0$) with other sets of plaintexts $\mathcal{L}_i, (2 \leq i \leq 8)$, selected like $\mathcal{L}_1$, in order to construct the datasets $\mathcal{D}_i$, $(1 \leq i \leq 8)$ using the 30-round characteristic that has been built using $\mathbf{RK}_i$, $(1 \leq i \leq 8)$ in order to get 4 right pairs per each dataset.

3.5.2 Key Recovery

We first prepare 24 groups of counters in which each group consists of 16 counters. Each group corresponds to a nibble of the key involved in the analysis. After that, we perform the attack in three sequential stages as follows.

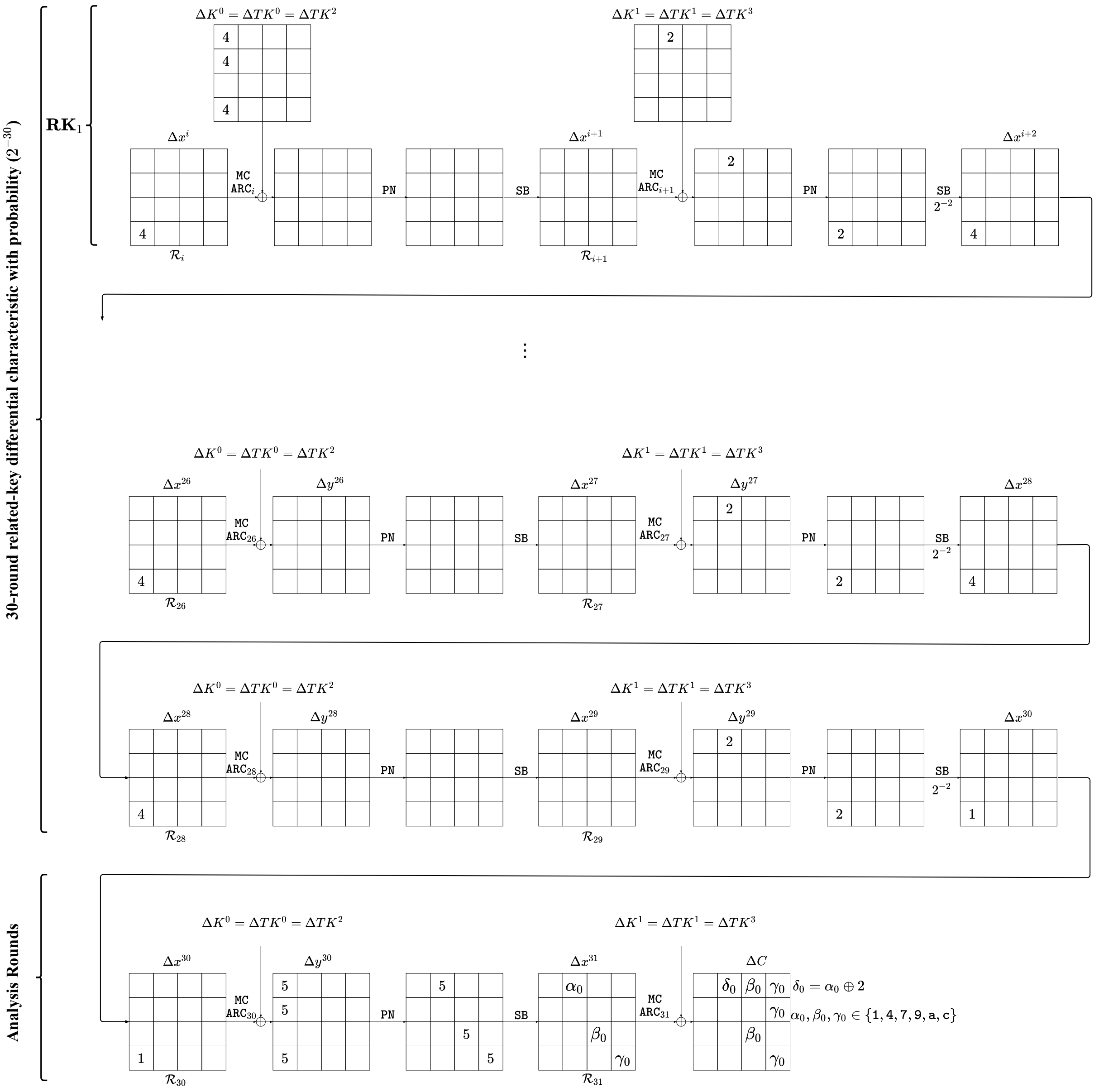

Figure 3.5: Related-key differential attack against Full CRAFT using the dataset $(\mathcal{D}_1)$ to recover $K^1_1, K^1_{10}, K^1_{15}$, and K^0_{12}.

First Stage. In this stage, we manage to determine the nibbles $K^1_i, (8 \leq i \leq 15)$. For example, we determine the right value of K^1_{15} as follows. We consider the group of counters corresponding to K^1_{15}, then for each right pair in the datasets $\mathcal{D}_1$ and $\mathcal{D}_5$, we guess K^1_{15} and decrypt the ciphertext nibble (C_{15}) (see Figures 3.5 and 3.6), then increment the counter corresponding to the guessed value if the difference $\Delta y^{30}_0 = 5$. After repeating

Table 3.3: Key recovery

Key Nibble	Dataset Used	Key Nibble	Dataset Used
K_0^0	$\mathcal{D}_{13}$	K_0^1	$\mathcal{D}_4$
K_1^0	$\mathcal{D}_{14}$	K_1^1	$\mathcal{D}_1$
K_2^0	$\mathcal{D}_{15}$	K_2^1	$\mathcal{D}_2$
K_3^0	$\mathcal{D}_{16}$	K_3^1	$\mathcal{D}_3$
K_4^0	$\mathcal{D}_9$	K_4^1	$\mathcal{D}_7$
K_5^0	$\mathcal{D}_{10}$	K_5^1	$\mathcal{D}_6$
K_6^0	$\mathcal{D}_{11}$	K_6^1	$\mathcal{D}_5$
K_7^0	$\mathcal{D}_{12}$	K_7^1	$\mathcal{D}_8$
K_8^0	$\mathcal{D}_5$	K_8^1	$\mathcal{D}_3$
K_9^0	$\mathcal{D}_6$	K_9^1	$\mathcal{D}_2$
K_{10}^0	$\mathcal{D}_7$	K_{10}^1	$\mathcal{D}_1$
K_{11}^0	$\mathcal{D}_8$	K_{11}^1	$\mathcal{D}_4$
K_{12}^0	$\mathcal{D}_1$	K_{12}^1	$\mathcal{D}_2$, $\mathcal{D}_6$
K_{13}^0	$\mathcal{D}_2$	K_{13}^1	$\mathcal{D}_3$, $\mathcal{D}_7$
K_{14}^0	$\mathcal{D}_3$	K_{14}^1	$\mathcal{D}_4$, $\mathcal{D}_8$
K_{15}^0	$\mathcal{D}_4$	K_{15}^1	$\mathcal{D}_1$, $\mathcal{D}_5$

these steps for all the pairs, we select the value corresponding to the highest counters as the right value for K_{15}^1.

By repeating these steps, we are able to obtain the right values of the nibbles $K_i^1, (8 \leq i \leq 15)$. Table 3.3 summarizes which datasets are used to recover these nibbles.

Second Stage. After finishing the first stage, we have the right value of the key nibbles $K_8^1, K_9^1, K_{10}^1, K_{11}^1, K_{12}^1, K_{13}^1, K_{14}^1, K_{15}^1$. During this stage, we obtain the right value of another 8 nibbles $K_0^1, K_1^1, K_2^1, K_3^1, K_{12}^0, K_{13}^0, K_{14}^0, K_{15}^0$. To this end, we consider, for example, the groups of counters corresponding to the key nibbles K_1^1 and K_{12}^0, respectively. After that, we reuse the dataset $\mathcal{D}_1$ (see Figure 3.5) in order to carry out the following steps:

1. Use the key nibbles K_9^1 and K_{13}^1 determined in the first stage to partially decrypt the ciphertext nibbles (C_9, C_{13}) and obtain the values of the nibbles x_9^{31} and x_{13}^{31}, respectively.

2. Guess K_1^1 and partially decrypt the ciphertext nibble C_1 to get the value and the difference at y_{12}^{30}, after that, increment the counter corresponding to the value of K_1^1 in case of $\Delta y_{12}^{30} = 5$.

3. Determine the right value of the key nibble K_1^1 by observing the highest counter.

4. Guess K_{12}^0 and decrypt y_{12}^{30} to get the difference Δy_1^{29}, then increment the counter corresponding to the value of K_{12}^0 if $\Delta y_1^{29} = 2$.

5. Determine the right value of the key nibble K_{12}^0 by observing the highest counter.

In the same manner, we reuse the datasets $\mathcal{D}_2, \mathcal{D}_3$ and $\mathcal{D}_4$ to determine the right values of the key nibbles (K_2^1, K_{13}^0), (K_3^1, K_{14}^0), (K_0^1, K_{15}^0), respectively.

Third Stage. Similar to the second stage, we reuse the datasets $\mathcal{D}_5, \mathcal{D}_6$, $\mathcal{D}_7$ and $\mathcal{D}_8$ to recover the key nibbles $K_i^1, (4 \leq i \leq 7)$ and $K_j^0, (8 \leq j \leq 11)$ as follows. To recover the nibbles K_6^1 and K_8^0, we consider the groups of counters corresponding them, and we reuse the dataset $\mathcal{D}_5$ (see Figure 3.6) in order to carry out the following steps:

1. Use the key nibble K_{14}^1 determined in the first stage to partially decrypt the ciphertext nibbles (C_{14}) to obtain the value of the nibble x_{14}^{31}.

2. Guess K_6^1 and get the value and the difference at y_8^{30}, then increment the counter corresponding to the value of K_6^1 in case of $\Delta y_8^{30} = 5$.

3. Determine the right value of the key nibble K_6^1 by observing the highest counter.

4. Guess K_8^0 and decrypt y_8^{30} to get the difference Δy_6^{29}, then increment the counter corresponding to the value of K_8^0 if $\Delta y_6^{29} = 2$.

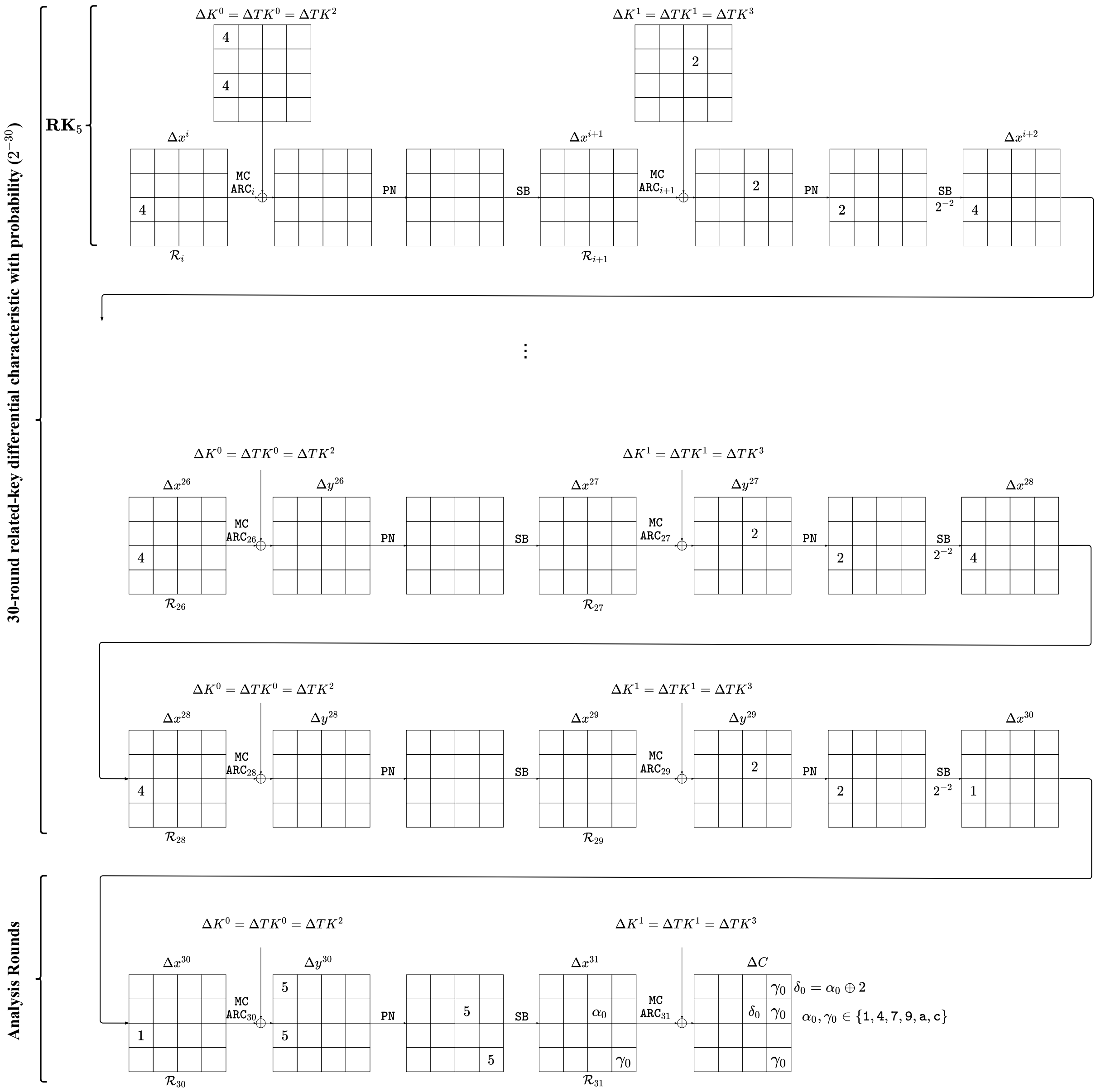

Figure 3.6: Related-key differential attack against Full CRAFT using the dataset ($\mathcal{D}_5$) to recover K_6^1, K_{15}^1, and K_8^0.

5. Determine the right value of the key nibble K_6^0 by observing the highest counter.

Using the same approach, we are able to determine the right values of the key nibbles (K_5^1, K_9^0), (K_4^1, K_{10}^0) and (K_7^1, K_{11}^0) using the datasets $\mathcal{D}_6, \mathcal{D}_7$ and $\mathcal{D}_8$, respectively.

3.5.3 Attack Complexity

Each set of plaintexts $\mathcal{L}_0, \ldots, \mathcal{L}_8$ contains 2^{32} plaintexts. Thus, we need $9 \times 2^{32} \approx 2^{35.17}$ queries to the encryption oracle.

During the first stage of the key recovery phase, we determine 4 nibbles using 32 right pairs and another 4 nibbles using 16 right pairs, therefore, we execute $2 \times (32+16) \times 2^4 = 2^{10.58}$ single nibble encryptions. For the second stage, we recover another 8 nibbles using 4 right pairs per each nibble. This process needs $2 \times 4 \times 4 \times (2 + 2^4 + 2^4) = 2^{10.08}$ single nibble encryptions. The third stage needs $2 \times 4 \times 4 \times (1 + 2^4 + 2^4) = 2^{10.04}$ single nibble encryptions. Therefore, these three stages need $2^{12.32}$ single nibble encryptions which is equivelant to $2^{11.83} \times \frac{1}{16} \times \frac{1}{32} \approx 8$ full-round encryptions. After these stages, we run exhaustive search over the remaining 2^{32} keys using one plaintext/ciphertext pair and this step needs 2^{32} full-round encryptions.

The dominant part of the memory complexity of these stages is for storing $4 \times 8 = 32$ right pairs in addition to the 128-bit right key. Therefore, the memory complexity is $2 \times 32 + 2 = 66$ 64-bit blocks.

3.5.4 Omitting the Exhaustive Search Step

In this section, we describe how we can omit the exhaustive search over 2^{32} keys. To this end, we utilize the repeatable 2-round characteristics $\mathbf{RK}_9 - \mathbf{RK}_{16}$ to build another 8 30-round characteristics. Then, we employ these characteristics to construct the datasets $\mathcal{D}_9 - \mathcal{D}_{16}$ to get, on average, 4 right pairs per each dataset as we do before.

To determine the right value of the key nibbles $K_i^0, (0 \leq i \leq 7)$, we first prepare 16 counters per each nibble. Then, we partially decrypt some nibbles of the ciphertexts. After that, we guess the key nibble and increment the counters if a specific nibble at the state y^{29} has a difference equal to 2, as we do in the second and the third stages before.

In this case, we need $17 \times 2^{32} \approx 2^{36.09}$ queries to the encryption oracle. In addition to the 8 full-round encryptions required during the previous three stages, we need $2 \times 4 \times 4 \times (6 + 2^4) = 2^{9.46}$ single nibble encryptions to recover the nibbles $K_0^0 - K_3^0$ and $2 \times 4 \times 4 \times (4 + 2^4) = 2^{9.32}$ single nibble encryptions to recover the nibbles $K_4^0 - K_7^0$. Thus, we need $8 + ((2^{9.46} + 2^{9.32}) \times \frac{1}{16} \times \frac{1}{32}) \approx 11$ full-round encryptions. Also, we need more $2 \times 4 \times 8 = 64$ block of memory to store the right pairs. Thus, the total memory complexity will be $66 + 64 = 130$ blocks of memory.

3.6 Summary

In this chapter, we studied the security of the lightweight tweakable block cipher `CRAFT` against the related-key differential cryptanalysis. More precisely, we described a systematic method to build a repeatable 2-round related-key differential characteristic that holds with the probability of 2^{-2}. We utilized this method to build several 30-round related-key differential characteristics with probability 2^{-30}. Then, we employed these characteristics to mount a key recovery attack against the full round of `CRAFT` in practical time. Moreover, we have verified this attack experimentally.

Chapter 4

Impossible Differential Attack on Reduced-Round SPARX-128/256

SPARX-128/256 is a member of the SPARX-128 block cipher family [28]. It has 128-bit block size and 256-bit key size. SPARX has been developed using ARX-based S-boxes with the aim of achieving provable security against single-trail differential and linear cryptanalysis. In this chapter, we propose two 20-round impossible differential distinguishers for SPARX-128. Then, we utilize these distinguishers to attack 24 rounds (out of 40 rounds) of SPARX-128/256. Our attack has time complexity of 2^{232} memory accesses, memory complexity of $2^{160.81}$ 128-bit blocks, and data complexity of 2^{104} chosen plaintexts.

4.1 Introduction

SPARX is a new family of ARX-based block ciphers proposed by Dinu *et al.* [28] at Asiacrypt 2016. To guarantee provable security against single-characteristic differential and linear cryptanalysis, the authors utilized a new design strategy, called the long trail design strategy. SPARX-128/256 has been analyzed using integral and multidimensional zero-correlation attacks [28, 99]. The designers reported that the best attack against

SPARX-128/256 is an integral attack based on the division property [95]. The attack covers 24 rounds with time, memory, and data complexities of 2^{233}, 2^{202}, 2^{104}, respectively. In this chapter, we present an impossible differential attack, which also covers 24 rounds, but has better time and memory complexities compared to the reported integral attack. Applying the following techniques allowed us to improve the attack complexity:

1. Instead of guessing the round keys, we utilize pre-computation tables in order to filter the round keys lead to the impossible differential distinguisher. A straightforward construction of the pre-computation tables at the S-box level would also render the time complexity of the attack to exceed the exhaustive search complexity. To overcome this problem, we consider the cascaded S-boxes in the same branch as a one large S-box and construct 3 look-up tables corresponding to the 3 branches involved in the analysis phase.

2. To overcome the problem of requiring to guess the whole master key in order to evaluate the 4 analysis rounds, we utilized a specific difference at the beginning of our distinguishers, which enables us to bypass the last S-box of branch 2, in the top 4 analysis rounds, with probability 1. Consequently, we need to guess only seven 32-bit words of the master key to test our distinguishers.

3. We utilize two distinguishers concurrently, which allows us to reduce the data complexity by a factor of two and also enhance the time complexity of the attack.

The remaining of this chapter is organized as follows. In Section 4.2, we provide a brief description of SPARX-128. Then, the details of the two impossible differential distinguishers are presented in Section 4.3. Afterwards, we present our key recovery attack in Section 4.4. Finally, the chapter is summarized in Section 4.5.

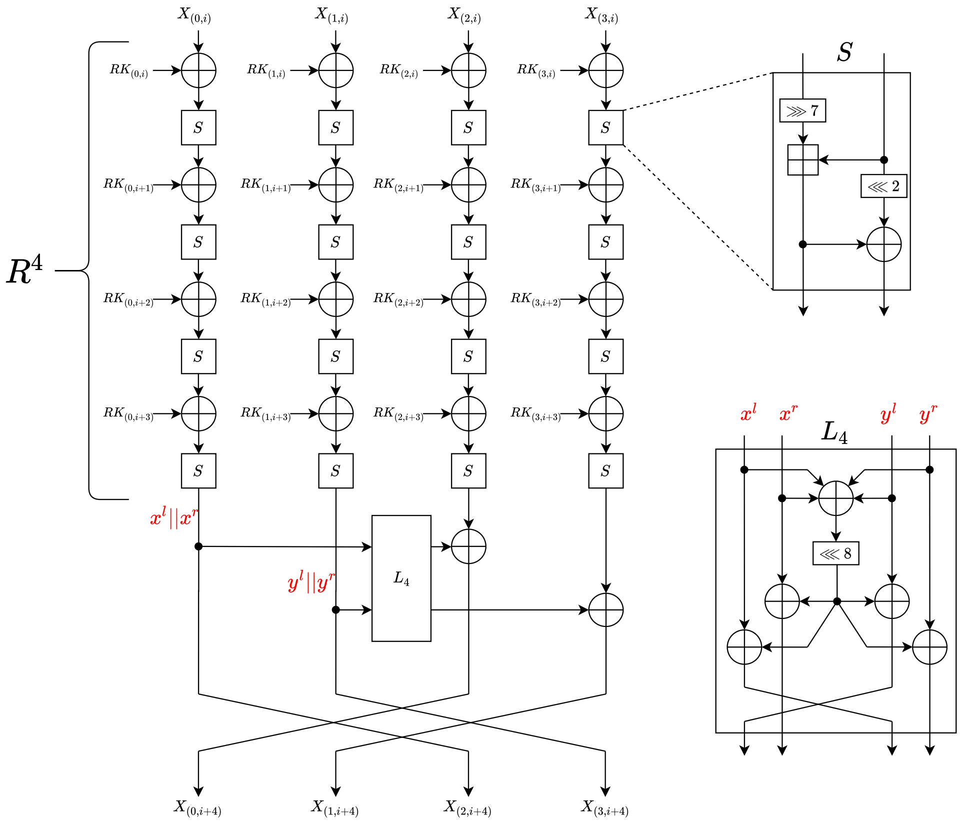

Figure 4.1: SPARX-128 step structure where $RK_{(a,i)}$ is the round key used at round i of branch a.

4.2 Description of SPARX-128

Let K_i denote the i^{th} 32-bit of the key state, $0 \leq i \leq 7$. Also, let $X_{(a,i)}$ denote the 32-bit input at branch a of round i, and $0 \leq a \leq 3$, where $a = 0$ corresponding to the left branch. The iteration of 4 rounds of SPECKEY [5] with their corresponding key additions is denoted by R^4. SPARX-128 is designed with the concept of iterating a big block called step. As depicted in Figure 4.1, each step has four parallel branches followed by a linear mixing layer (L_4) which is applied to ensure diffusion between the branches. The branch structure is made of four cascaded ARX-based S-boxes that interleave with key addition (R^4). The structure made of a key addition followed by an S-box in the four parallel branches is called a round. For further details, the reader is referred to [28].

4.3 Impossible Differential Distinguishers of SPARX

In this section, we present two 20-round impossible differential distinguishers for SPARX-128. As shown in Figure 4.2, the first distinguisher starts at round i with only branch 3 ($X_{(3,i)}$) having a nonzero difference Δ_0. By propagating this difference one step forward and by utilizing the property of the linear mixing layer ($L_4(0,0) = (0,0)$), only branch 1 ($X_{(1,i+4)}$) has a nonzero difference Δ_1. After propagating this difference one more step through R^4 and L_4, the values of $X_{(0,i+8)}$, $X_{(1,i+8)}$, and $X_{(2,i+8)}$ will be Δ_3, Δ_4, and 0, respectively. From the other direction (at round $i+20$), we choose branches 0, 1, 3 to have the nonzero differences α_2, α_1, and α_0, respectively, such that $L_4(0,\alpha_0) = (\alpha_2,\alpha_1)$. These chosen differences allow us to pass two steps (8 rounds) backward with only branch 3 ($X_{(3,i+12)}$) having a nonzero difference α_4. Thus, branch 2 ($X_{(2,i+8)}$) has zero difference in the forward direction which contradicts with the backward direction where it has a nonzero difference α_5 after applying the linear layer $L_4(0,\alpha_4) = (\alpha_5,\alpha_6)$ and before applying the inverse of R^4. Moreover, there is another contradiction at branch 0. The second distinguisher can be constructed by utilizing the forward path of the first distinguisher, and in the backward path by choosing branches 0, 1, 2 to have nonzero differences β_2, β_1, and β_0, respectively, such that $L_4(\beta_0,0) = (\beta_2,\beta_1)$.

4.4 Impossible Differential of SPARX-128/256

The attack is constructed by appending one step (4 rounds) on the top of the two 20-round distinguishers as depicted in Figure 4.3. Based on the key schedule of SPARX-128/256, the 256-bit master key is divided into 8 32-bit keys denoted by K_i, $0 \le i \le 7$. Then, the round keys of the first step are extracted as stated in Figure 4.3 where $R(K_i)$ denotes the output of applying the round operation (S-box only) to K_i. In our attack, we use a specific difference at the beginning of our distinguishers, which enables us to bypass

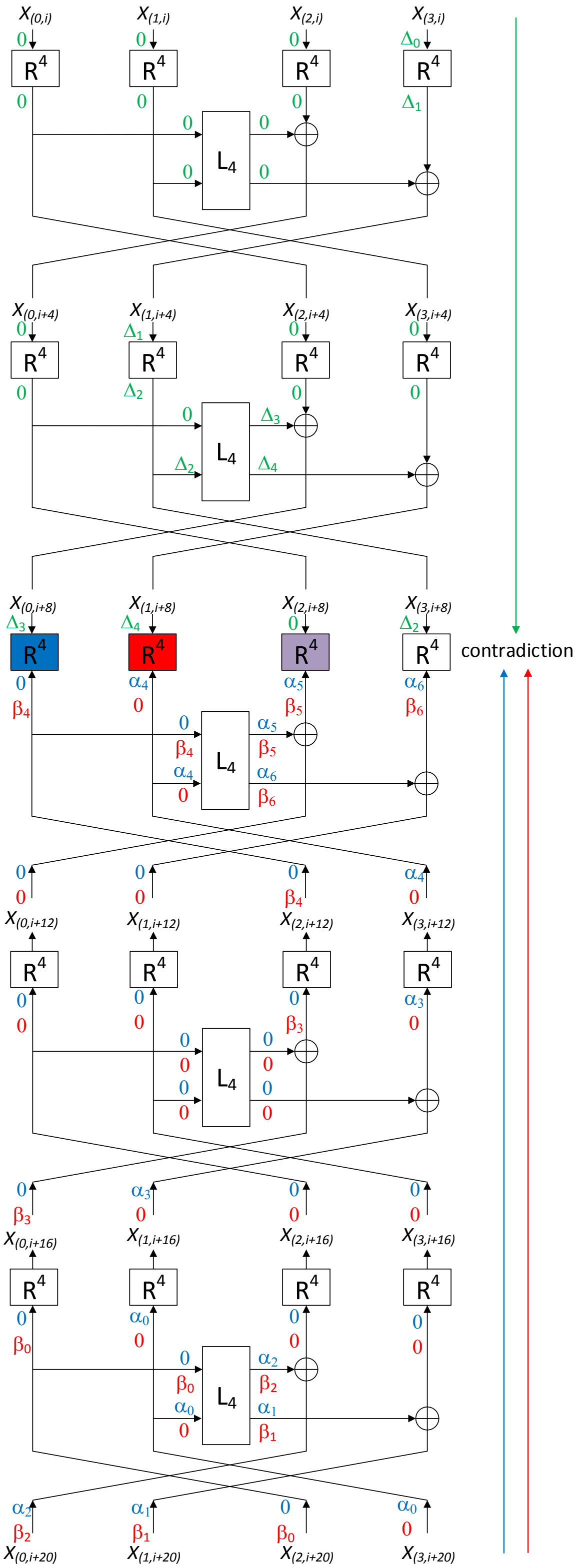

Figure 4.2: Two 20-round impossible differential distinguishers of SPARX-128. Both distinguishers have the same forward path. The blue and red colors in the backward direction correspond to the first and second distinguishers, respectively.

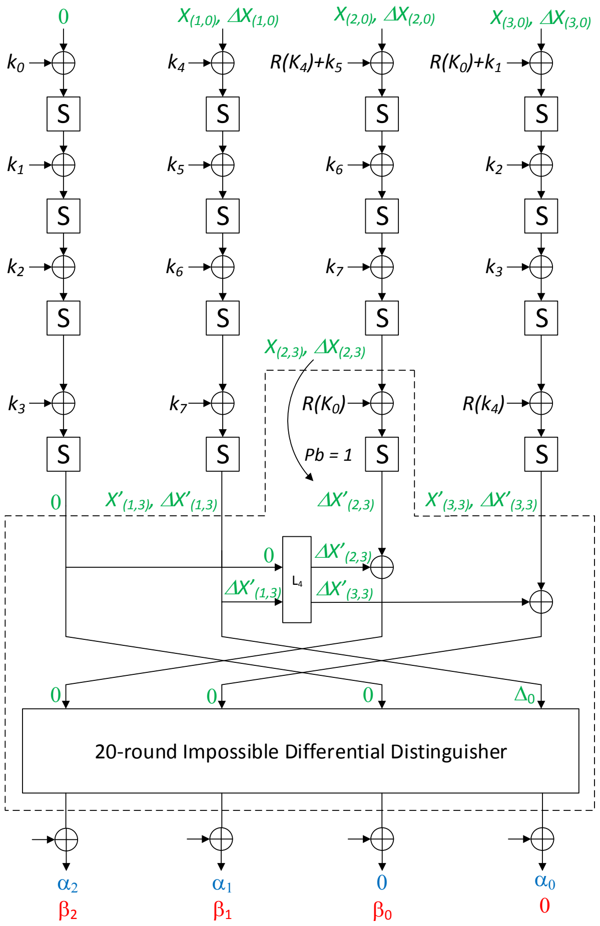

Figure 4.3: A 24-round impossible differential cryptanalysis of SPARX-128/256, where $X'_{(a,i)} = S(X_{(a,i)} \oplus RK_{(a,i)})$ and $RK_{(a,i)}$ is the round key used at round i of branch a.

the last S-box of branch 2, in the top 4 analysis rounds, with probability 1, see Figure 4.3. The values of $(\Delta X'_{(1,3)}$, $\Delta X_{(2,3)}$, $\Delta X'_{(3,3)})$ can be chosen from the following set:

$$\begin{aligned}\{&(0x00000080, 0x00400000, 0x80008080),\\ &(0x00020202, 0x00408000, 0x00020200),\\ &(0x00020282, 0x00008000, 0x80028280)\}.\end{aligned} \tag{4.1}$$

These values are calculated using the following procedure. Firstly, the value of $\Delta X_{(2,3)}$ is chosen to exploit the fact that there exist differentials with probability 1 for one SPECKEY round $(Prob.[\Delta X_{(2,3)} \rightarrow \Delta X'_{(2,3)}] = 1)$. After that, the value of $\Delta X'_{(2,3)}$ is used as a fixed point for L_4 to obtain the values of $\Delta X'_{(1,3)}$ and $\Delta X'_{(3,3)}$ such that $L_4(0, \Delta X'_{(1,3)}) =$

$(\Delta X'_{(2,3)}, \Delta X'_{(3,3)})$. The attack has two phases; the pre-computation and the online phases.

4.4.1 Pre-computation Phase

In this phase, we create 3 hash tables (H_1, H_2, H_3) corresponding to branches 1, 2, and 3 that are involved in the analysis rounds as shown in Figure 4.3. These tables will be used to extract/filter the wrong keys during the key recovery step of the online phase. These pre-computation tables are created, by assigning ($\Delta X'_{(1,3)}$, $\Delta X_{(2,3)}$, $\Delta X'_{(3,3)}$) one of the three tuples given in Equation (4.1), and then proceeding as follows:

- H_1: For the chosen $\Delta X'_{(1,3)}$ and for all $2^{32\times 5=160}$ possible values of $X'_{(1,3)}$, K_4, K_5, K_6, and K_7, we obtain the corresponding values of $X_{(1,0)}$ and $\Delta X_{(1,0)}$. Then, we store the values of $K_4||K_5||K_6||K_7$ in the table indexed by the values of $X_{(1,0)}$ and $\Delta X_{(1,0)}$. As a result, we have $2^{160}/2^{64} = 2^{96}$ values for $K_4||K_5||K_6||K_7$ per row.

- H_2: We initialize H_2 with the binary value 0. Then, for the chosen $\Delta X_{(2,3)}$ and for all $2^{32\times 4}$=2^{128} possible values of $X_{(2,3)}$, $R(K_4) + K_5$, K_6, and K_7, we obtain $X_{(2,0)}$ and $\Delta X_{(2,0)}$. Then, for each computed value, we store the binary value 1 in the table indexed by the values of $X_{(2,0)}$, $\Delta X_{(2,0)}$, $R(K_4) + K_5$, K_6, and K_7. Here, the binary values 0 and 1 denote invalid and valid entries, respectively. Consequently, we have one valid entry every 2^{32} entries.

- H_3: For the chosen $\Delta X'_{(3,3)}$ and for all $2^{32\times 5=160}$ possible values of $X'_{(3,3)}$, $R(K_0)+K_1$, K_2, K_3, and $R(K_4)$, we obtain $X_{(3,0)}$ and $\Delta X_{(3,0)}$. Then, we store the values of $R(K_0)+K_1||K_2||K_3$ in the table indexed by the values of $X_{(3,0)}$, $\Delta X_{(3,0)}$, and $R(K_4)$. Thus we have $2^{160}/2^{96} = 2^{64}$ values of keys $R(K_0) + K_1||K_2||K_3$ per row.

4.4.2 Online Phase

In this phase, we collect a set of plaintext/ciphertext pairs. Then, we utilize the pre-computation tables to discard wrong keys, and then recover the correct master key.

Data Collection. We choose 2^n structure of plaintexts at round 0 such that branch 0 has a fixed value and the other three branches vary over all the possible values. Hence, each structure includes about $2^{32\times3=96}$ plaintexts and can generate $2^{96}\times(2^{96}-1)/2\approx 2^{191}$ plaintext pairs. Therefore, we have $2^n \times 2^{191} = 2^{n+191}$ pairs of plaintexts in total. After that, we query the encryption oracle and keep only the plaintext pairs whose ciphertext differences match the pattern $\alpha_2, \alpha_1, 0, \alpha_0$ such that $L_4(0,\alpha_0) = (\alpha_2, \alpha_1)$ or the pattern $\beta_2, \beta_1, \beta_0, 0$ such that $L_4(\beta_0, 0) = (\beta_2, \beta_1)$. The probability that one of these patterns occurs is about $2^{-3\times32} + 2^{-3\times32} = 2^{-95}$. The expected number of remaining pairs after this filtration is about $2^{n+191-95} = 2^{n+96}$.

Key Recovery. For each plaintext pair obtained in the data collection, we apply the following procedure:

1. Determine all possible values of keys $K_4||K_5||K_6||K_7$ that satisfy the value of $\Delta X'_{(1,3)}$ by accessing the table H_1 with the value of the plaintext pairs. The expected number of returned 128-bit keys is 2^{96}.

2. Deduce the value of $R(K_4)+K_5$, then access the table H_2 with value of the plaintext pairs. The expected number of the remaining 128-bit keys after this filtration is $2^{96-32} = 2^{64}$.

3. For each 128-bit key remaining from the above step, deduce the value of $R(K_4)$, then access table H_3 to obtain $R(K_0)+K_1||K_2||K_3$ that satisfy the value of $\Delta X'_{(3,3)}$. Thus, the expected number of these keys is 2^{64}. As a result, and after analyzing one plaintext pair, the number of 224-bit keys which lead to the impossible differentials (wrong keys) is $2^{64+64} = 2^{128}$.

4. After analyzing all the pairs and discarding all 224-bit wrong keys (only one correct candidate remains), we exhaustively search for the 32-bit K_0 and deduce the

value of K_1 from the value of $R(K_0) + K_1$. Then, we test these values using 2 plaintext/ciphertext pairs to obtain the correct master key.

Attack Complexity. For each one of the 2^{n+96} plaintext pairs remaining after the ciphertext filtration, we discard, on average, 2^{128} out of 2^{224} possible values of the keys. Therefore, the probability that a wrong key is not discarded after analyzing one pair is $1 - 2^{128-244} = 1 - 2^{-96}$. Thus, after analyzing all pairs, there are $2^{224} \times (1 - 2^{-96})^{2^{n+96}} \approx 2^{224} \times (e^{-1})^{2^{n+96-96}} \approx 2^{224} \times 2^{-1.4 \times 2^n}$ remaining candidates of 224-bit keys. To have only the correct 224-bit key, we choose $n = 8$ to satisfy $2^{224} \times 2^{-1.4 \times 2^n} \leq 1$. Therefore, the data complexity is $2^{n+96} = 2^{104}$ chosen plaintext. The memory complexity is dominated by the pre-computation phase. Table H_1 has 2^{160} entries $\times (4 \times 32)$ bits per entry $= 2^{167}$ bits $= 2^{160}$ 128-bit blocks. Table H_2 has 2^{160} entries $\times 1$ bit per entry $= 2^{160}$ bits $= 2^{153}$ 128-bit blocks. Table H_3 has 2^{160} entries $\times (3 \times 32)$ bits per entry $\approx 2^{166.58}$ bits $= 2^{159.58}$ 128-bit blocks. Thus, the memory complexity is $2^{160} + 2^{153} + 2^{159.58} \approx 2^{160.81}$ 128-bit blocks. The time complexity of the offline phase , which is the time required to build the pre-computation tables, is $2 \times (2^{160} \times 4 + 2^{128} \times 3 + 2^{160} \times 4) \times \frac{1}{24 \times 4} \approx 2^{157.42}$ 24-round encryption. The time complexity of the online phase is dominated by the memory accesses during the key recovery which is $2^{104+96} + 2^{104+96} + 2^{104+64+64} \approx 2^{232}$ memory access. In addition, there is the time of the exhaustive search of the correct key which requires $2 \times 2^{32} = 2^{33}$ 24-round encryption.

4.5 Summary

In this chapter, we investigated the security of SPARX-128/256 against the impassible differential cryptanalysis. Precisely, We presented two 20-round impossible differential distinguishers for SPARX-128. Accordingly, we utilized them to attack 24 rounds of SPARX-128/256. The data, memory access time, and memory complexities of the attack are 2^{104}, 2^{232}, and $2^{160.81}$, respectively.

Chapter 5

On MILP-Based Automatic Search for Differential Trails Through Modular Additions with Application to Bel-T

Using modular addition as a source of nonlinearity is frequently used in many symmetric-key structures such as ARX and Lai–Massey schemes [62]. At FSE'16, Fu *et al.* [41] proposed a Mixed Integer Linear Programming (MILP)-based method to handle the propagation of differential trails through modular additions assuming that the two inputs to the modular addition and the consecutive rounds are independent. However, this assumption does not necessarily hold. In this chapter, we study the propagation of the XOR difference through the modular addition at the bit level and show the effect of the carry bit. Then, we propose a more accurate MILP model to describe the differential propagation through the modular addition taking into account the dependency between the consecutive modular additions. The proposed MILP model is utilized to launch a differential attack against Bel-T-256, which is a member of the Bel-T block cipher family

[10] that has been adopted recently as a national standard of the Republic of Belarus.

5.1 Introduction

As we mentioned in Chapter 2, different optimization techniques such as Mixed Integer Linear Programming (MILP) attracted the attention of many cryptanalysis researchers. For differential cryptanalysis, Sun *et al.* [91] proposed MILP models that represent exactly the propagation of the differential through SPN block ciphers in order to automate the search for high probability (related-key) differential or linear characteristics. One of the downsides of these MILP models was the inability to efficiently describe the Difference Distribution Table (DDT) of large (8-bit) S-boxes which was tackled by Abdelkhalek *et al.* [1].

Regarding ARX-based block ciphers, Fu *et al.* [41] represented the conditions developed by Lipmaa and Moriai [65] (hereafter referred to as Lipmaa's conditions) by a set of MILP constraints in order to automate the search for the best differential trail through the modular addition. In this representation, the authors assume that the two inputs to modular addition and the consecutive component of the cipher's round function are independent. However, this assumption is very often not satisfied, especially with round functions that have two or more consecutive modular operations [103]. In the same context, Leurent [64] provides a tool based on finite state machines to automate the search for differential characteristics through the modular addition considering the constraints due to several consecutive bits of the modular addition inputs. However, the complexity of this analysis is linear in the number of states, and the number of states can be exponential in the size of the system, which according to the authors, makes this approach suitable only to study systems with a limited number of states.

In this chapter, we revisit the conditions stated by Lipmaa and Moriai [65] to verify the possibility of an XOR difference of two inputs of addition modulo 2^n to produce a specific XOR difference at the output. In particular, we deduce the conditions on the bits of the inputs and the output of addition modulo 2^n that have to be satisfied in order to propagate an XOR difference of the inputs to a particular XOR difference at the output. Using these conditions, we describe some examples showing that using Lipmaa's conditions with the independence assumption between the consecutive components of a block cipher is not enough to ensure the validity of the derived differential characteristic. To address this problem, we propose a new MILP model considering the dependency between two or more successive modular additions.

To illustrate the effectiveness of our approach, we apply our method to attack the block cipher Bel-T, which is a family of block ciphers that has been approved as the national standard of the Republic of Belarus [10], formerly known by its Russian name Belorussia. The Bel-T family includes three block ciphers, denoted as Bel-T-k, all of them have the same block size of 128 bits and a variable key length (k) of 128, 192 or 256 bits. The designers of Bel-T combined a Lai-Massey scheme [62] with a Feistel network [39] to build a complex round function with 7 S-box layers per round. The round function is iterated 8 times to construct the whole cipher. Concretely, we employ our MILP approach beside a Hamming weight-based partial DDT to search for a differential distinguisher for Bel-T. Then, we mount a $4\frac{1}{7}$-round differential attack on round-reduced Bel-T-256 which, up to our knowledge, is the best published attack against this cipher in the single-key setting. Moreover, we show that the Bel-T block cipher is not a Markov cipher [63], *i.e.,* the validity of the differential characteristic depends on the used secret key. In this context, we also provide a systematic method to define the set of keys that can be attacked using our differential characteristic.

Few cryptanalysis results on Bel-T block ciphers have been published including fault-based attacks [53] and the related-key differential attack on round-reduced Bel-T-256 [2]. It should be noted that in the related-key differential attack presented in [2], the modular addition is modeled using the method proposed by Fu *et al.* [41] with the independency assumption. We verified the distingusiher presented in [2] and found it to be invalid as it involves two modular additions that share the same input and have conflicting condition. In addition, we will report in Chapter 6 our integral attacks on ($3\frac{2}{7}$ and $3\frac{6}{7}$)-round reduced Bel-T-256 in the single-key setting. These integral attacks was published in [34]. Table 5.1 contrasts our attack with the integral attacks in [34].

The rest of this chapter is organized as follows. In Section 5.2, we briefly revisit the XOR differential characteristic of modular addition. The developed MILP-based method, which is used to search for the differential characteristic, is explained in Section 5.3. In Section 5.4, we describe how we apply the new MILP model to find a differential distinguisher for Bel-T. Then, the details of our attack are presented in Section 5.5. Finally, the chapter is concluded in Section 5.6.

Table 5.1: Attack results on Bel-T-256

Model	Attack	#Rounds	Data	Time	Memory	Reference
Single Key	Integral	$3\frac{2}{7}$	2^{13}	$2^{199.33}$	-	[34]
		$3\frac{6}{7}$	2^{33}	$2^{254.61}$	-	[34]
	Differential	$4\frac{1}{7}$	2^{114}	$2^{237.14}$	2^{224}	Sec. 5.5

5.2 XOR-Differential Characteristics of Modular Addition

Definition 5.1 *Let α, β and γ be fixed n-bit XOR differences. The XOR-differential probability (DP) of addition modulo 2^n (xdp^+) is the probability with which α and β propagate to γ through the modular addition operation, computed over all pairs of n-bit inputs (x,y):*

$$xdp^+(\alpha, \beta \to \gamma) = 2^{-2n} \times \#\{(x, y) : ((x \oplus \alpha) \boxplus (y \oplus \beta)) \oplus (x \boxplus y) = \gamma\}.$$

Lipmaa and Moriai [65] stated the following two conditions that have to be satisfied in order for the XOR input differences (α, β) to propagate to an output difference (γ) through the addition modulo 2^n:

1. The bit-wise XOR of the least significant bit of the inputs and output differences must be 0, *i.e.,* $\alpha_0 \oplus \beta_0 \oplus \gamma_0 = 0$ which is equivalent to $\gamma_0 = \alpha_0 \oplus \beta_0$.

2. If the three bits $\alpha_i, \beta_i,$ and γ_i are equal, then the XOR of the subsequent bits $\alpha_{i+1}, \beta_{i+1},$ and γ_{i+1} must equal these bits as well, *i.e.,* $\alpha_{i+1} \oplus \beta_{i+1} \oplus \gamma_{i+1} = \alpha_i = \beta_i = \gamma_i$ for $0 \leq i \leq n-2$.

If these two conditions above are satisfied, then the probability of the differential characteristic (xdp^+) can be calculated as:

$$xdp^+(\alpha, \beta \rightarrow \gamma) = 2^{-\sum_{i=0}^{n-2} \neg eq(\alpha_i, \beta_i, \gamma_i)}$$

where $\neg eq$ is 0 when $(\alpha_i, \beta_i, \gamma_i)$ are the same, and 1 otherwise. By using these conditions, we can determine if a differential characteristic $(\alpha, \beta \rightarrow \gamma)$ is a valid one or not. For example, the characteristic $(\alpha, \beta \rightarrow \gamma) = (0001, 0001 \rightarrow 0001)$ is impossible because it breaks the first condition.

In the remaining of this section, we show our interpretation of these two conditions by deriving the relationship between the input and output differences at the bit level.

Let $\boldsymbol{x} = (x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$[1], $\boldsymbol{y} = (y_{n-1}, y_{n-2}, \ldots, y_1, y_0)$, and $\boldsymbol{z} = (z_{n-1}, z_{n-2}, \ldots, z_1, z_0)$ be n-bit vectors where $\boldsymbol{z} = \boldsymbol{x} \boxplus \boldsymbol{y}$. Then, z_i can be iteratively expressed as follows:

$$z_0 = x_0 \oplus y_0 \oplus c_0, \qquad c_0 = 0, \tag{5.1}$$

$$z_{i+1} = x_{i+1} \oplus y_{i+1} \oplus c_{i+1}, \qquad c_{i+1} = x_i y_i \oplus x_i c_i \oplus y_i c_i \quad \forall i = 0, 1, \ldots, n-2. \tag{5.2}$$

[1] We use little-endian representation where x_0 is the least significant bit.

It is obvious that the Lipmaa's conditions are based on equations (5.1) and (5.2). Consider that we have two pairs $(\boldsymbol{x}, \boldsymbol{x}^*)$ and $(\boldsymbol{y}, \boldsymbol{y}^*)$ such that $\Delta\boldsymbol{x} = \boldsymbol{x} \oplus \boldsymbol{x}^*$, and $\Delta\boldsymbol{y} = \boldsymbol{y} \oplus \boldsymbol{y}^*$. The relation between the XOR input differences $\Delta\boldsymbol{x}, \Delta\boldsymbol{y}$ and the XOR output difference $\Delta\boldsymbol{z} = \boldsymbol{z} \oplus \boldsymbol{z}^*$ can be derived as follows: Let $\Delta\boldsymbol{x} = (\delta x_{n-1}, \delta x_{n-2}, \ldots, \delta x_1, \delta x_0)$, $\Delta\boldsymbol{y} = (\delta y_{n-1}, \delta y_{n-2}, \ldots, \delta y_1, \delta y_0)$, and $\Delta\boldsymbol{z} = (\delta z_{n-1}, \delta z_{n-2}, \ldots, \delta z_1, \delta z_0)$ be the XOR difference where $\delta x_i = x_i \oplus x_i^*$, $\delta y_i = y_i \oplus y_i^*$, and $\delta z_i = z_i \oplus z_i^*$, respectively. The Lipmaa's first condition comes from equation (5.1) in which $\delta z_0 = \delta x_0 \oplus \delta y_0 \oplus \delta c_0$, but $\delta c_0 = 0$ as $c_0 = c_0^* = 0$. Therefore, for $(\Delta\boldsymbol{x}, \Delta\boldsymbol{y} \to \Delta\boldsymbol{z})$ to be a possible differential characteristic, the relation $(\delta z_0 = \delta x_0 \oplus \delta y_0)$ must be satisfied.

For given input and output differences at two successive bits $((\delta x_i, \delta y_i, \delta z_i)$ and $(\delta x_{i+1}, \delta y_{i+1}, \delta z_{i+1}))$, we can use equation (5.2) to calculate the XOR difference at the carry bit δc_{i+1} using the following two equations:

$$\begin{aligned}\delta c_{i+1} &= c_{i+1} \oplus c_{i+1}^* \\ &= x_i y_i \oplus x_i c_i \oplus y_i c_i \oplus x_i^* y_i^* \oplus x_i^* c_i^* \oplus y_i^* c_i^*, \qquad (5.3)\\ \delta c_{i+1} &= \delta z_{i+1} \oplus \delta x_{i+1} \oplus \delta y_{i+1} \qquad (5.4)\end{aligned}$$

To have a valid differential characteristic, the value of δc_{i+1} evaluated from these two equations must be consistent. For example, if we have $\delta x_i = \delta y_i = \delta z_i = 0$, this implies that $\delta c_i = 0$, *i.e.*, if $x_i^* = x_i$, $y_i^* = y_i$, $z_i^* = z_i$ then $c_i^* = c_i$. Therefore, from equation (5.3), $\delta c_{i+1} = 0$. Consequently, $\delta z_{i+1} \oplus \delta x_{i+1} \oplus \delta y_{i+1} = 0$ must hold with probability 1.

As another example, let us consider the following XOR differences: $\delta x_i = \delta y_i = 0$, and $\delta z_i = 1$, this implies that $\delta c_i = 1$, *i.e.*, if $x_i^* = x_i$, $y_i^* = y_i$ and $z_i^* = \bar{z}_i$ then $c_i^* = \bar{c}_i$ where $\bar{z}_i$, $\bar{c}_i$ are the bit-wise NOT of z_i, c_i, respectively. As a result, the value of δc_{i+1} from equation (5.3) will depend on the relation between x_i and y_i as follows: $\delta c_{i+1} = x_i \oplus y_i$. If δc_{i+1} is 0, then the condition $x_i = y_i$ must be satisfied. In this case, from equation (5.2), the output bit z_i will equal to c_i and the carry bit c_{i+1} will be equal to x_i.

By iterating over all possible values of $\delta x_i, \delta y_i, \delta z_i$ and δc_{i+1}, we can drive the conditions on the bits x_i, y_i, z_i, c_i and c_{i+1} to have a valid differential characteristic. We summarize these conditions in Table 5.2, in which the condition column is divided into three sub-columns: the first one is the direct condition similar to the one we derived in the previous examples. The second and third sub-columns are the values of z_i and c_{i+1} in case the direct condition, the first sub-column, is satisfied.

It should be noted that Lipmaa's second condition is specified by the first two rows and last two rows of Table 5.2, *i.e.*, if $\delta x_i, \delta y_i$ and δz_i are equal, then $\delta c_{i+1} = \delta z_{i+1} \oplus \delta x_{i+1} \oplus \delta y_{i+1}$ has to equal them.

Table 5.2: Relation between δx_i, δy_i, δz_i and δc_{i+1}

δz_i	δy_i	δx_i	δc_i	δc_{i+1}	Condition		
0	0	0	0	0	No condition		
0	0	0	0	1	Invalid		
0	0	1	1	0	$x_i = \bar{c}_i$	$z_i = \bar{y}_i$	$c_{i+1} = y_i = \bar{z}_i$
0	0	1	1	1	$x_i = c_i$	$z_i = y_i$	$c_{i+1} = x_i = c_i$
0	1	0	1	0	$y_i = \bar{c}_i$	$z_i = \bar{x}_i$	$c_{i+1} = x_i = \bar{z}_i$
0	1	0	1	1	$y_i = c_i$	$z_i = x_i$	$c_{i+1} = y_i = c_i$
0	1	1	0	0	$x_i = \bar{y}_i$	$z_i = \bar{c}_i$	$c_{i+1} = c_i = \bar{z}_i$
0	1	1	0	1	$x_i = y_i$	$z_i = c_i$	$c_{i+1} = x_i = y_i$
1	0	0	1	0	$x_i = y_i$	$z_i = c_i$	$c_{i+1} = x_i = y_i$
1	0	0	1	1	$x_i = \bar{y}_i$	$z_i = \bar{c}_i$	$c_{i+1} = c_i = \bar{z}_i$
1	0	1	0	0	$y_i = c_i$	$z_i = x_i$	$c_{i+1} = y_i = c_i$
1	0	1	0	1	$y_i = \bar{c}_i$	$z_i = \bar{x}_i$	$c_{i+1} = x_i = \bar{z}_i$
1	1	0	0	0	$x_i = c_i$	$z_i = y_i$	$c_{i+1} = x_i = c_i$
1	1	0	0	1	$x_i = \bar{c}_i$	$z_i = \bar{y}_i$	$c_{i+1} = y_i = \bar{z}_i$
1	1	1	1	0	Invalid		
1	1	1	1	1	No condition		

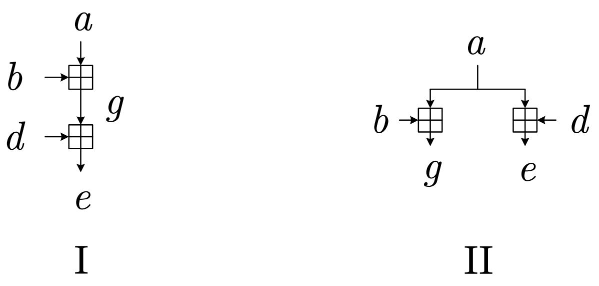

Figure 5.1: Examples of incompatible conditions

5.2.1 Examples of Incompatible Conditions

In this section, we show some examples in which using Lipmaa's conditions with the independency assumption between the consecutive components of the block cipher is not enough to ensure the validity of the differential characteristic.

Example 1: Consider the two cascaded modular operations shown in Figure (5.1.I) and the following XOR differences:

$$\Delta\boldsymbol{a} = 00000001 \qquad \boldsymbol{g} = \boldsymbol{a} \boxplus \boldsymbol{b} \qquad\qquad \boldsymbol{e} = \boldsymbol{g} \boxplus \boldsymbol{d}$$

$$\Delta\boldsymbol{b} = 00000000 \qquad \Delta\boldsymbol{g} = 00001111 \qquad \Delta\boldsymbol{d} = 00000000 \qquad \Delta\boldsymbol{e} = 00001101$$

When looking at each modular addition operation individually, each one satisfies the Lipmaa's conditions and holds with probability 2^{-4}. Assuming independency, the whole differential characteristic should hold with probability 2^{-8}, however, it is actually an impossible characteristic. To explain, using Table 5.2, we can show that if the characteristic holds for the first operation, $\boldsymbol{g} = (g_{n-1}, \cdots, g_1, g_0)$ will have a specific pattern ($g_1 = g_0$) due to the carry effect. On the other hand, the characteristic will hold for the second modular addition if $\boldsymbol{g}$ has a specific pattern ($g_1 = \bar{g}_0$), also due to the carry effect.

To further explain this carry effect, consider for the first operation the differences of the first three bits $(\delta g_0, \delta b_0, \delta a_0) = (1, 0, 1)$, $(\delta g_1, \delta b_1, \delta a_1) = (1, 0, 0)$ and $(\delta g_2, \delta b_2, \delta a_2) = (1, 0, 0)$. We access Table 5.2 twice with $(\delta z_i,\ \delta y_i,\ \delta x_i,\ \delta c_i,\ \delta c_{i+1}) = (\delta g_0, \delta b_0, \delta a_0,\ \delta c_0,\ \delta c_1)$ = (1,0,1,0,1) where the carry $\delta c_0 = \delta g_0 \oplus \delta b_0 \oplus \delta a_0$ and the carry $\delta c_1 = \delta g_1 \oplus \delta b_1 \oplus \delta a_1$, and with $(\delta z_i, \delta y_i, \delta x_i, \delta c_i, \delta c_{i+1}) = (\delta g_1, \delta b_1, \delta a_1,\ \delta c_1,\ \delta c_2) = (1{,}0{,}0{,}1{,}1)$ where the carry $\delta c_2 = \delta g_2 \oplus \delta b_2 \oplus \delta a_2$. From the first access, we get the following condition:

$$b_0 = \bar{c}_0 \Rightarrow g_0 = \bar{a}_0 \text{ and } c_1 = a_0 = \bar{g}_0 \tag{5.5}$$

And from the second access, we get the condition:

$$a_1 = \bar{b}_1 \Rightarrow g_1 = \bar{c}_1 \text{ and } c_2 = c_1 = \bar{g}_1 \tag{5.6}$$

From equation (5.5), if the characteristic is valid for the first bit, the carry bit c_1 will equal to $\bar{g}_0$. Also, if the characteristic is valid for the second bit, the same carry bit c_1 will have a relation with g_1 as determined by equation (5.6). By combining these two relations, we prove that the output $\boldsymbol{g}$ has the pattern $(g_1 = g_0)$.

Using the same methodology, we can also prove that the characteristic will hold for the second operation if the input $\boldsymbol{g}$ has the pattern $(g_1 = \bar{g}_0)$ which contradicts with the output of the first operation. All these patterns have also been verified experimentally.

Example 2: Let us consider another ordering of two modular operations as shown in Figure (5.1.II) and the following XOR differences:

$$\begin{array}{cccc}
\Delta\boldsymbol{a} = 00001111 & \boldsymbol{g} = \boldsymbol{a} \boxplus \boldsymbol{b} & & \boldsymbol{e} = \boldsymbol{a} \boxplus \boldsymbol{d} \\
\Delta\boldsymbol{b} = 00000001 & \Delta\boldsymbol{g} = 00010000 & \Delta\boldsymbol{d} = 00000001 & \Delta\boldsymbol{e} = 00000000
\end{array}$$

Again, the two operations individually satisfy the Lipmaa's conditions. However, the first operation requires the input $\boldsymbol{a}$ to be in a specific pattern $(a_0 = a_1 = a_2 = a_3)$ and the second operation requires the input $\boldsymbol{a}$ to be in another contradicting pattern $(a_0 = a_1 = a_2 = \bar{a}_3)$.

5.3 New MILP Model for Differential Characteristics of Modular Addition

Fu *et al.* [41] represent Lipmaa's conditions by a set of MILP constraints in order to automate the search for the best differential trail through the modular addition. As explained in the previous section, Lipmaa's conditions are not enough to ensure the validity of the derived differential characteristic especially when the block cipher structure has two or more consecutive modular additions. We propose a more accurate MILP model to automate the search for differential characteristics through modular additions taking into account the dependency between two consecutive modular additions that put more constraints on the values of input and output bits.

In order to represent the relation between two consecutive bits i and $i-1$ on a variable $\boldsymbol{x}$, we define a new variable called $x_i^{\oplus} = x_i \oplus x_{i-1}$ which can take a value of $\{0, 1, ?\}$; it is set to 0 if the condition $x_i = x_{i-1}$ is required and set to 1 if the condition $x_i = \bar{x}_{i-1}$ is required. Also, $x_i^{\oplus}$ can be kept undetermined (?) which means it can be 0 or 1 if there is no restriction on the relation between x_i and x_{i-1}.

Evaluation of $(z_i^{\oplus}, y_i^{\oplus}, x_i^{\oplus})$ for a modular addition. The relation between the bits x_i and x_{i-1}, for the input $\boldsymbol{x}$ in a modular addition comes through the carry bit c_i. Therefore the variable $x_i^{\oplus}$ can be evaluated as:

$$x_i^{\oplus} = (x_i \oplus c_i) \oplus (c_i \oplus x_{i-1})$$

where $x_i \oplus c_i$ and $c_i \oplus x_{i-1}$ can take a value of $\{0, 1, ?\}$ like $x_i^{\oplus}$ and the bit-wise XOR of ? with any value equals to ?. Based on Table 5.2, the values of $(x_i \oplus c_i)$ and $(c_i \oplus x_{i-1})$ reflect the situation where there are conditions that should be satisfied to get the XOR differences $(\delta z_i, \delta y_i, \delta x_i, \delta c_{i+1})$ and $(\delta z_{i-1}, \delta y_{i-1}, \delta x_{i-1}, \delta c_i)$, respectively. Thus, the values of

$(z_i^{\oplus}, y_i^{\oplus}, x_i^{\oplus})$ will be determined based on the XOR differences $(\delta z_{i-1}, \delta y_{i-1}, \delta x_{i-1}, \delta z_i, \delta y_i, \delta x_i, \delta c_{i+1})$. We develop Algorithm 1 to determine these values. The input of our proposed algorithm is a general-purpose data structure dictionary $\mathbb{D}$ which is obtained by reformatting the valid rows in Table 5.2 where the relations between the current bits (z, y, x) with the current carry bit c and the subsequent carry bit $c_{_{+1}}$ are derived from the condition column in Table 5.2 and indexed by the value of the XOR difference of these bits, see Table 5.3. The output of Algorithm 1 is the truth table $\mathbb{T}$ of $(z_i^{\oplus}, y_i^{\oplus}, x_i^{\oplus})$ as a function of the possible XOR differences $(\delta z_{i-1}, \delta y_{i-1}, \delta x_{i-1}, \delta z_i, \delta y_i, \delta x_i, \delta c_{i+1})$. Out of $2^7 = 128$ values of these bits, there are only 98 values that can be used as possible differences. Table 5.4 shows part of the derived truth table $\mathbb{T}$.

MILP constraints for Modular Addition. To automate the process of the search for the differential characteristic using MILP technique, we have to transform the truth table $\mathbb{T}$ into a set of linear constraints. To this end, we represent the rows of $\mathbb{T}$ combined with the value of $\neg eq(\delta z_i, \delta y_i, \delta x_i)$ as a set of points in 11-dimensional binary vector space by substituting ? with all possible values, *e.g.,* the row (0010010??1) associated with $\neg eq(0, 0, 1) = 1$ will be described by 4 binary vectors: (00100100011, 00100100111, 00100101011, 00100101111). After this step, we have 640 binary vectors which have a convex hull. We use the `inequality_generator()` function in Sage[2] to obtain the H-Representation which is a set of linear inequalities that describe the vectors of this convex hull. We can use this set of inequalities as MILP constraints to present the possible XOR differences in two successive bits $(\delta z_{i-1}, \delta y_{i-1}, \delta x_{i-1}, \delta z_i, \delta y_i, \delta x_i)$ and the carry of the third bits (δc_{i+1}) combined with the conditions on the value of these bits represented as $(z_i^{\oplus}, y_i^{\oplus}, x_i^{\oplus})$. In our case, the number of generated inequalities is 313, which is very large to be handled by any MILP optimizer. Therefore, we employ the Greedy algorithm proposed by Sun *et al.* in [91] to reduce this set to only 24 inequalities. In

[2] http://www.sagemath.org/

order to link the current bit with the following bits, we encoded equation (5.4), which is a bit-wise XOR of three inputs and one output, by 8 linear inequalities utilizing the truth table of the bit-wise XOR and `inequality_generator()` function in Sage. In this manner, we have represented the relation between three successive bits using $24 + 8 = 32$ inequalities and this representation is repeated for $i = 1, 2, \ldots, n-2$. In order to complete the MILP modeling for the modular addition, we describe the condition on the first bit $(i = 0)$ $\delta z_0 \oplus \delta y_0 \oplus \delta x_0 = 0$ associated with $\neg eq(\delta z_0, \delta y_0, \delta x_0)$ by 4 linear inequalities. Accordingly, we can represent the difference propagation through the addition modulo 2^n taking into account the relation between the value of two successive bits using $32 \times (n-2) + 4$ inequalities. The objective function of the MILP optimizer would minimize $\sum_{i=0}^{n-2} \neg eq(\delta z_i, \delta y_i, \delta x_i)$, which denotes the log_2 probability of the underlying characteristic.

Algorithm 1: Truth table generator

```
Input  : The Dictionary $\mathbb{D}$.
Output: The truth table $\mathbb{T}$ of $(z_i^{\oplus}, y_i^{\oplus}, x_i^{\oplus})$ as a function of the possible XOR
        differences $(\delta z_{i-1}, \delta y_{i-1}, \delta x_{i-1}, \delta z_i, \delta y_i, \delta x_i, \delta c_{i+1})$
begin
    $\mathbb{T} = \emptyset$
    for $2^7$ possible values of $(\delta z_{i-1}, \delta y_{i-1}, \delta x_{i-1}, \delta z_i, \delta y_i, \delta x_i, \delta c_{i+1})$ do
        $\delta c_{i-1} \leftarrow \delta z_{i-1} \oplus \delta y_{i-1} \oplus \delta x_{i-1}$
        $\delta c_i \leftarrow \delta z_i \oplus \delta y_i \oplus \delta x_i$
        if $(\delta z_{i-1}, \delta y_{i-1}, \delta x_{i-1}, \delta c_{i-1}, \delta c_i)$ in $\mathbb{D}$.keys AND $(\delta z_i, \delta y_i, \delta x_i, \delta c_i, \delta c_{i+1})$ in
         $\mathbb{D}$.keys then
            RCarry1 $\leftarrow \mathbb{D}[(\delta z_i, \delta y_i, \delta x_i, \delta c_i, \delta c_{i+1})][0]$
            RCarry2 $\leftarrow \mathbb{D}[(\delta z_{i-1}, \delta y_{i-1}, \delta x_{i-1}, \delta c_{i-1}, \delta c_i)][1]$
            $(z_i^{\oplus}, y_i^{\oplus}, x_i^{\oplus}) \leftarrow$ RCarry1 $\oplus$ RCarry2
            $\mathbb{T} \leftarrow \mathbb{T} \cup \{(\delta z_{i-1}, \delta y_{i-1}, \delta x_{i-1}, \delta z_i, \delta y_i, \delta x_i, \delta c_{i+1}, z_i^{\oplus}, y_i^{\oplus}, x_i^{\oplus})\}$
        end
    end
    return $\mathbb{T}$
end
```

Table 5.3: The dictionary $\mathbb{D}$.

$\mathbb{D}$.keys					$\mathbb{D}[*][0]$	$\mathbb{D}[*][1]$
δz	δy	δx	δc	δc_{+1}	$c \oplus (z,y,x)$	$c_{+1} \oplus (z,y,x)$
0	0	0	0	0	(? , ? , ?)	(? , ? , ?)
0	0	1	1	0	(? , ? , 1)	(1 , 0 , ?)
0	0	1	1	1	(? , ? , 0)	(? , ? , 0)
0	1	0	1	0	(? , 1 , ?)	(1 , ? , 0)
0	1	0	1	1	(? , 0 , ?)	(? , 0 , ?)
0	1	1	0	0	(1 , ? , ?)	(1 , ? , ?)
0	1	1	0	1	(0 , ? , ?)	(? , 0 , 0)
1	0	0	1	0	(0 , ? , ?)	(? , 0 , 0)
1	0	0	1	1	(1 , ? , ?)	(1 , ? , ?)
1	0	1	0	0	(? , 0 , ?)	(? , 0 , ?)
1	0	1	0	1	(? , 1 , ?)	(1 , ? , 0)
1	1	0	0	0	(? , ? , 0)	(? , ? , 0)
1	1	0	0	1	(? , ? , 1)	(1 , 0 , ?)
1	1	1	1	1	(? , ? , ?)	(? , ? , ?)

Table 5.4: Part of the truth table $\mathbb{T}$.

δz_{i-1}	δy_{i-1}	δx_{i-1}	δz_i	δy_i	δx_i	δc_{i+1}	$z_i^{\oplus}$	$y_i^{\oplus}$	$x_i^{\oplus}$
				....					
0	0	1	0	0	1	0	?	?	1
0	0	1	0	0	1	1	?	?	0
0	0	1	0	1	1	0	0	?	?
0	0	1	0	1	1	1	1	?	?
0	0	1	1	0	1	0	?	0	?
0	0	1	1	0	1	1	?	1	?
0	1	0	0	1	0	0	?	1	?
0	1	0	0	1	0	1	?	0	?
0	1	0	0	1	1	0	0	?	?
0	1	0	0	1	1	1	1	?	?
0	1	0	1	1	0	0	?	?	0
0	1	0	1	1	0	1	?	?	1
0	1	1	0	0	1	0	?	?	1
0	1	1	0	0	1	1	?	?	0
0	1	1	0	1	0	0	?	1	?
0	1	1	0	1	0	1	?	0	?
0	1	1	0	1	1	0	0	?	?
				....					

5.4 Application to Bel-T

5.4.1 Bel-T Specification

Since the official Bel-T specification is available only in Russian, we rely on the English version of the specification that is provided by Jovanovic and Polian, who presented fault-based attacks on the Bel-T block cipher family [53]. Bel-T has a 128-bit block size and a variable key length of 128, 192 or 256 bits. The 128-bit plaintext P is split into 4 32-bit words, *i.e.*, $P = A_0^0||B_0^0||C_0^0||D_0^0$. The round function of Bel-T consists of 7 S-box layers in which a 32-bit mapping function (G_r) is combined with one or two modulo operations as illustrated in Figure 5.2. Then, this round function is repeated 8 times for all versions of Bel-T. The function G_r (G-box) maps a 32-bit word $w = w_1||w_2||w_3||w_4$, with $w_i \in \{0,1\}^8$, as follows: $G_r(w) = (H(w_1)||H(w_2)||H(w_3)||H(w_4)) \lll r$. Here, H is an 8-bit S-box and $\lll r$ denotes left shift rotation by r positions ($r \in \{5, 13, 21\}$). The specification of the 8-bit S-box can be found in [53].

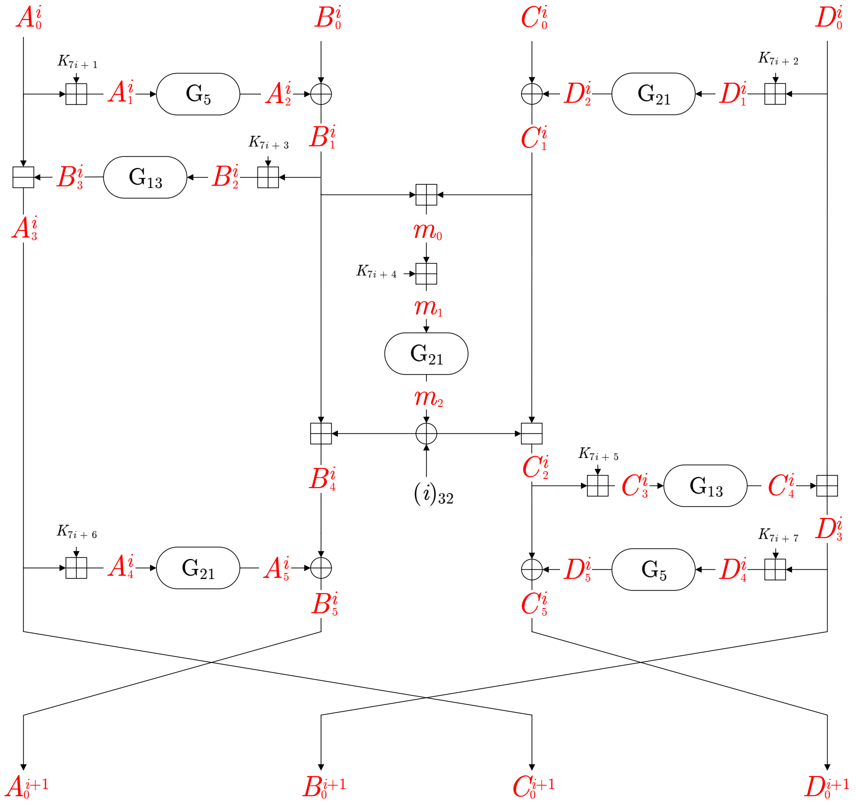

Figure 5.2: Bel-T round function. $\oplus, \boxplus, \boxminus$ denote bit-wise XOR, arithmetic addition and subtraction modulo 2^{32} respectively, and $(i)_{32}$ denotes the round number represented as 32-bit word.

Key Schedule. In all versions of Bel-T, the 128-bit plaintext block P is encrypted using a 256-bit encryption key denoted as $K_1||\dots||K_8$, where K_i is a 32-bit word for $1 \leq i \leq 8$. The encryption key is distributed among the round keys as shown in Table 5.5. The encryption key is extracted from the master key as follows:

- Bel-T-256: the encryption key is identical to the master key.

- Bel-T-192: the master key is formatted as $K_1||\dots||K_6$ and K_7, K_8 are set to $K_7 := K_1 \oplus K_2 \oplus K_3$ and $K_8 := K_4 \oplus K_5 \oplus K_6$.

- Bel-T-128: the master key is formatted as $K_1||\dots||K_4$ and K_5, K_6, K_7, K_8 are set to $K_5 := K_1$, $K_6 := K_2$, $K_7 := K_3$ and $K_8 := K_4$.

Table 5.5: Encryption key schedule of Bel-T, where i and K_{7i+j} denote the round number and the round key, respectively.

i	K_{7i+1}	K_{7i+2}	K_{7i+3}	K_{7i+4}	K_{7i+5}	K_{7i+6}	K_{7i+7}
0	K_1	K_2	K_3	K_4	K_5	K_6	K_7
1	K_8	K_1	K_2	K_3	K_4	K_5	K_6
2	K_7	K_8	K_1	K_2	K_3	K_4	K_5
3	K_6	K_7	K_8	K_1	K_2	K_3	K_4
4	K_5	K_6	K_7	K_8	K_1	K_2	K_3
5	K_4	K_5	K_6	K_7	K_8	K_1	K_2
6	K_3	K_4	K_5	K_6	K_7	K_8	K_1
7	K_2	K_3	K_4	K_5	K_6	K_7	K_8

5.4.2 MILP-based Search for Differential Characteristic of Bel-T

To search for differential characteristics in a block cipher using MILP, the difference propagation through its components is described using a set of linear constraints. In Bel-T, this means generating a set of linear inequalities to describe how an XOR difference would propagate through a bit-wise XOR, an addition/subtraction modulo 2^{32}, and an 8-bit S-box. As the difference propagates with probability through the non-linear components, its associated probability is incorporated in the corresponding linear inequalities. The objective function of the MILP model would be to maximize this probability, which we do by minimizing the negative of the base-2 logarithm of this probability.

Bit-wise XOR. If $\delta x_i, \delta y_i$ and δz_i represent the bit-level differences, then the difference propagation through the bit-wise XOR operation $\delta x_i \oplus \delta y_i = \delta z_i$ can be represented by 5 linear inequalities [91]. Using the truth table of the XOR operation, these can be further reduced to the following 4 linear inequalities:

$$\delta x_i + \delta y_i - \delta z_i \geq 0,\ \delta x_i - \delta y_i + \delta z_i \geq 0,\ -\delta x_i + \delta y_i + \delta z_i \geq 0,\ -\delta x_i - \delta y_i - \delta z_i \geq -2.$$

Modular Addition and Subtraction. We use the new MILP model described in Section 5.3 to propagate the input differences $(\Delta\boldsymbol{x}, \Delta\boldsymbol{y})$ to an output difference $(\Delta\boldsymbol{z})$ through the addition modulo 2^{32} such that $\boldsymbol{x} \boxplus \boldsymbol{y} = \boldsymbol{z}$ using $32 \times (32-2) + 4 = 964$ inequalities. Since the subtraction modulo 2^n, $\boldsymbol{x} \boxminus \boldsymbol{y} = \boldsymbol{z}$ is equivalent to $\boldsymbol{x} = \boldsymbol{y} \boxplus \boldsymbol{z}$, the difference propagation through modular subtraction can be described in a similar way as that used to describe modular addition.

Modular Addition with a Secret Key. The Bel-T round function encompasses a modular addition with a secret key which has zero difference in a single-key differential attack. This operation can then be expressed as $\boldsymbol{x} \boxplus \boldsymbol{k} = \boldsymbol{z}$ and the differential characteristic as $(\Delta\boldsymbol{x}, 0) \rightarrow \Delta\boldsymbol{z}$. Therefore, the difference propagation through this operation can be described in a similar way as that used to describe modular addition by inserting 32 more constraints to explicitly set $\Delta\boldsymbol{y} = 0$. The number of required constraints will be $964 + 32 = 996$. Indeed, we can improve this description by decreasing the number of MILP constraints to roughly half as follows. We repeat the steps described in Section 5.3 using the rows of the truth table $\mathbb{T}$ that have $\delta y_{i-1} = \delta y_i = 0$ and also $\delta y_{i+1} = 0$. Consequently, the number of MILP constraints decreases to $(13+4)(32-2)+2 = 512$.

8-bit S-box. Using the Sage `inequality_generator()` function to model the DDT of an 8-bit S-box is computationally infeasible. Therefore, the use of MILP to search for differential characteristics was restricted to block ciphers that do not include 8-bit S-boxes. Abdelkhalek *et al.* [1] have put forward an approach to model the DDT of an 8-bit S-box efficiently. First, the DDT is split into several tables corresponding to unique probability values. After assigning binary variables to each unique probability value, these binary variables are represented as Boolean functions in the input and output difference bits, *i.e.,* each boolean function is 1 when the input difference is propagated to the output difference with the corresponding probability value, and 0 otherwise. Next, the Quine-McCluskey algorithm [71, 78] was used to transform the Boolean functions to their reduced Product of Sum (PoS) which can then be described by a set of linear inequalities. To describe the deterministic propagation of the zero-difference, an additional binary variable was used as a sort of flag, *i.e.,* when it is 0, the S-box is inactive and therefore both the input and output differences are set to 0. When it is 1, the S-box is active and one probability value along with input difference and corresponding output difference are chosen. As in ARX block ciphers, the probability of the differential characteristic gets lower when more bits are active, we decided to follow the approach in [2] in which we do not use the high probability entries in the DDT, but rather the entries with low Hamming weight in the input and output differences. Throughout our experiments, we have limited the Hamming weight of the input and output difference not to exceed 3. However, the partial DDT was still too large to be handled directly using the `inequality_generator()` function and hence we augmented our approach with the approach proposed by Abdelkhalek *et al.* for handling the DDT of large S-boxes to describe the partial DDT using linear inequalities. Based on our implementation, $1,660$ linear inequalities are needed to describe this Hamming weight-based partial DDT.

Lai-Massey Scheme. Since the Lai-Massey scheme is invertible, the following constraints are added to our model to enforce the output of the Lai-Massey scheme (B_4^i, C_2^i) to be non-zero when its input (B_1^i, C_1^i) is non-zero, see Figure 5.2.

$$\begin{aligned}
\sum_{j=0}^{n-1} B_{1,j}^i + \sum_{j=0}^{n-1} C_{1,j}^i + LM_i &\geq 1, \\
\sum_{j=0}^{n-1} B_{4,j}^i + \sum_{j=0}^{n-1} C_{2,j}^i + 2n \times LM_i &\leq 2n, \\
\sum_{j=0}^{n-1} B_{1,j}^i + \sum_{j=0}^{n-1} C_{1,j}^i + 2n \times LM_i &\leq 2n, \\
\sum_{j=0}^{n-1} B_{4,j}^i + \sum_{j=0}^{n-1} C_{2,j}^i + LM_i &\geq 1.
\end{aligned}$$

In these constraints, LM_i is a dummy binary variable. If the input difference is zero, the first equation enforces LM_i to be 1 which enforces the output difference to be zero in the second equation. If the input difference is non-zero, the third equation enforces LM_i to be 0 which enforces the output difference to be non-zero in the fourth equation.

5.4.3 3-round Differential Characteristic

Using the above derived MILP model of the different components of the Bel-T, we are able to build a model of the whole round of Bel-T using $55,641$ linear inequalities and $2,647$ binary variables. Then, we used the Gurobi optimizer [48] on a server of two Xeon Processors E5-2697 ($2 \times 12 = 24$ cores in total) with 125 GB RAM to search for a differential characteristic of Bel-T. Consequently, we found a 2-round differential characteristic with probability 2^{-54} after about 4.5 hours. We use this characteristic as an initial solution for the optimizer in order to extend the characteristic to 3 rounds. After running the search process for 36 days, we were not able to find a 3-round differential characteristic better than the one that holds with probability 2^{-111}. The 3-round differential characteristic we use in our attack is shown in Figure 5.3 in which 0 denotes a 32-bit difference of all zeros, e_i, e_{i-j} and $e_{i,j,k,\ldots}$ denote 32-bit difference of all 0's and 1 at bit i, bits i to j, and bits $i, j, k, \ldots$, respectively.

5.4.4 Validity of The Differential Characteristic

In this section, we show that Bel-T block cipher is not a Markov cipher and the differential characteristic depends on the used secret key. Consequently, we propose a systematic way to obtain the ratio of the keys that can be attacked using our distinguisher.

Recall that a Markov cipher [63] is an iterated block cipher in which the probability of the difference, *e.g.,* the XOR difference through the individual operations of the round function is independent of the corresponding plaintext values of its input, if the round keys applied to each round are independent and chosen in a uniformly random manner.

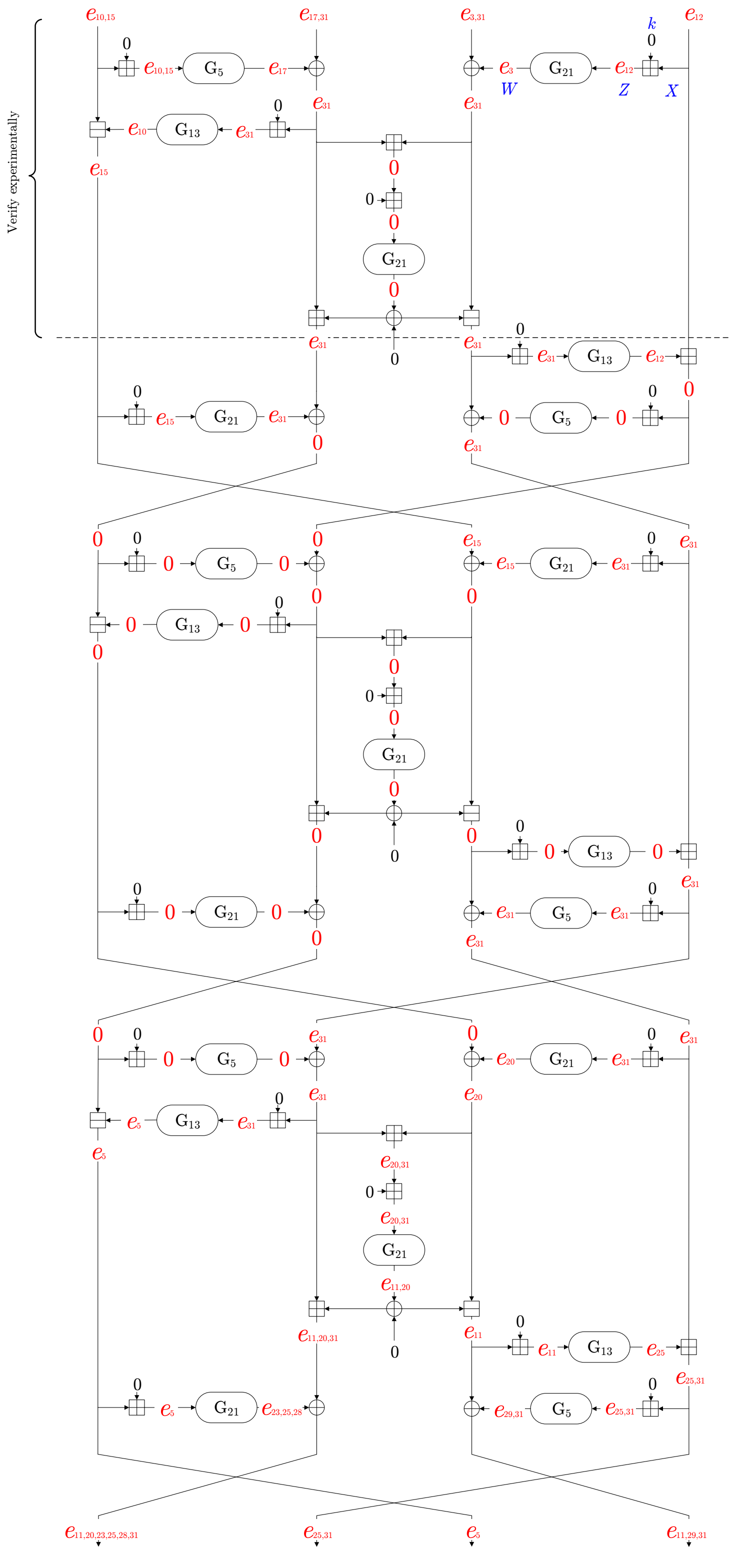

Figure 5.3: 3-round differential characteristic of Bel-T with probability 2^{-111}. 0 denotes a 32-bit difference of all zeros, e_i, e_{i-j} and $e_{i,j,k}$ denote a 32-bit difference of 0's and 1 at bit i, bits i to j, and bits i, j, and k, respectively

In the case of Bel-T, the secret key is mixed via modular addition operations, therefore the XOR difference propagation through these operations is probabilistic and depends on the used key. Additionally, the hypothesis of independent round keys does not hold due to the simple key schedule of Bel-T. Moreover, there are many two or more successive modular additions, which are not independent as shown in Section 5.2. For these reasons, we can conclude that Bel-T is not a Markov cipher.

Since the secret key is mixed via modular addition operations, Bel-T is not a *key-alternating* cipher [24] and the probability of the XOR difference of these modular operations may drop to zero due to the used key [16] and we therefore cannot use our distinguisher in this case. In the remaining of this section, we obtain the ratio of the keys (valid keys) which we can use the distinguisher with. We define the S-box layer to include the modular addition with a key followed by the G-box mapping (G_r). We consider a 32-bit key as an invalid key when the probability of the XOR difference through its S-box layer drops to zero independent of the other input of the modular addition.

Let us consider, *e.g.*, the S-box layer of K_2 in round 0 (see Figure 5.3) in which the key K_2 has a specific value k, $Z = X \boxplus k$ and $W = G_{21}(Z)$ where $\Delta X = \Delta Z = 0x00001000$, $\Delta k = 0x00000000$ and $\Delta W = 0x00000008$. Therefore, we are looking for the values of k that cannot give the output difference ΔW for any value of X.

For each value of k, we can exhaustively search over all possible values of the pair $(X, X \oplus \Delta X)$ to check if there is a value of X that leads to the output difference ΔW. If there is no such value, we consider k as invalid. The complexity of search for all possible values of K_2 will be roughly $\mathcal{O}(2^{64})$ which is computationally hard because we will repeat this search for all modular additions with keys.

Alternatively, we can obtain from Table 5.2 that the condition $k_{12} = c_{12}$, where k_{12} and c_{12} are the bit number 12 of the key and the carry respectively, is the only constraint that has to be checked to verify whether the key k is an invalid key or not. Also from the DDT of the G-box, the second byte of Z (bits from Z_8 to Z_{15}) in hexadecimal has to be one of $\{0x02, 0x12, 0x4C, 0x5C\}$ to satisfy the output difference ΔW. Accordingly, the following constraints have to be satisfied:

$$k_{12} = c_{12}, \qquad Z_8 = 0, \qquad Z_{13} = 0, \qquad Z_{15} = 0, \qquad \bar{Z}_9 = Z_{10} = Z_{11} = Z_{14}.$$

For each value of k, there is a value X that gives $Z_8 = 0$ with probability 1 because there are no conditions on k nor Z from bit 0 to 7. Given this fact and by using equations (5.1) and (5.2), we can prove that the carry bits $c_9 = c_{10} = c_{11} = c_{12} = 0$ if the key bits $k_8 = 1$ and $k_9 = k_{10} = k_{11} = 0$ independently of the corresponding bits of X. Therefore, if the key bit $k_{12} = 1$, the condition $k_{12} = c_{12}$ will be impossible. As a result, if the key k has the pattern $k_8 = k_{12} = 1$ and $k_9 = k_{10} = k_{11} = 0$, it will be an invalid key irrespective of the value X due to the contradiction between the two constraints $Z_8 = 0$ and $k_{12} = c_{12}$. We can manually search for such patterns but this process is very difficult, time-consuming, and error-prone.

Observation 5.1 *Consider a modular addition $\boldsymbol{z} = \boldsymbol{x} \boxplus \boldsymbol{y}$ where the bit z_i has a specific value. Then, the carry bit c_j (for $j > i$) depends on the input bits from i to $j - 1$ and is independent of the input bits from 0 to $i - 1$.*

The dependency between a carry bit c_j and the input bits from 0 to $j - 1$ is due to the carry chain (see equation 5.2). If we know that the output bit z_i has a specific value, we can evaluate the carry bit c_i as $c_i = z_i \oplus x_i \oplus y_i$ instead of evaluating it using the value of x_{i-1}, y_{i-1} and c_{i-1}. Thus, the carry chain and dependency are broken. Back to our example, given that $Z_8 = 0$, the carry bit c_{12} will depend on the bits from 8 to 11 of the inputs X and k based on the observation. Therefore, considering the key k as an invalid will depend on its bits from 8 to 12. In general, given a key k, if we exhaustively search over all possible values of the pair $(X,\ X \oplus \Delta X)$ and there is no value X that can lead to the difference ΔW, then the byte of the key containing the conditional bits is the reason for invalidating ΔW. We therefore can repeat the search for all possible value of these

Procedure Obtain Invalid Key Set

```
Input   : $\Delta X, \Delta W$
Output: $\mathbb{K}$
begin
    $\mathbb{K} = \emptyset$
    Determine PosOfBytes and NBytes which are the position and the number
      of bytes that have XOR difference in $\Delta X$
    for $2^{8\times\texttt{NBytes}}$ possible values of Bytes do
        Generate $k$ randomly such that the concatenation of the bytes in the
          position PosOfBytes has the value Bytes
        invalid = True
        for $2^{32}$ possible values of $X$ do
            if $G(X \boxplus k) \oplus G((X \oplus \Delta X) \boxplus k) = \Delta W$ then
                invalid = False
                break
            end
        end
        if invalid then
            $\mathbb{K} \leftarrow \mathbb{K} \cup \{\texttt{Bytes}\}$
        end
    end
    return $\mathbb{K}$
end
```

bytes. Consequently, the exhaustive search complexity in our example will be reduced roughly to $\mathcal{O}(2^{40})$ which is feasible.

The above approach can be generalized to determine the set of the byte values $\mathbb{K}$ leading to invalid keys as shown in Procedure (Obtain Invalid Key Set).

Table 5.6 summarizes the ratio of valid keys of each key K_i that has conditions in our distinguisher. It should be noted that the key K_2 is used in two rounds but the bytes that have the conditions are in different positions. Accordingly, the total ratio of the valid keys can be evaluated as the multiplication of all ratios of the valid keys which will be $2^{-3.8}$ corresponding to $2^{252.2}$ keys. In order to validate this result, we have experimentally verified the differential characteristic. In particular, we have opted the first four S-box layers of the differential characteristic of probability 2^{-24} (see Figure 5.3) and have found that the experimental probability matches on average the theoretical one for 4426 of 10000

Table 5.6: Ratio of valid keys

Round	Key	Ratio of valid keys
0	K_1	136/256
	K_2	216/256
	K_6	129/256
2	K_2	216/256
	K_3	144/256
	K_4	228/256
	K_5	192/256

Table 5.7: The difference at the points used in the attack

Point label	The difference in Binary			
A_0^3	10010010	10010000	00001000	00000000
B_0^3	10000010	00000000	00000000	00000000
C_0^3	00000000	00000000	00000000	00100000
D_0^3	10100000	00000000	00001000	00000000
B_1^3	???00000	000?????	????????	????????
C_1^3	????????	????????	???00000	000?????

randomly generated keys. Comparing with Table 5.6, this ratio is very close to the ratio of the valid keys for this part of the distinguisher.

5.5 Differential Attack on $4\frac{1}{7}$-Round Reduced Bel-T-256

In this section, we present a differential attack on $4\frac{1}{7}$-round reduced Bel-T-256 by appending one round and one S-box layer on the above derived differential distinguisher as illustrated in Figure 5.4. Our differential characteristic ends at A_0^3, B_0^3, C_0^3 and D_0^3 with values $e_{11,20,23,25,28,31}$, $e_{25,31}$, e_5 and $e_{11,29,31}$, respectively. Therefore, by propagating the differences at A_0^3 and D_0^3 through the S-box layers, we obtain the corresponding 32-bit difference at B_1^3 and C_1^3. Table 5.7 summarizes the difference in Binary at some points that we will use during the attack. Our attack has two phases: pre-computation phase and an online phase.

5.5.1 Pre-computation Phase

In this phase, we create 4 hash tables (H_1, H_2, H_3, H_4) corresponding to the S-box layers shown in Figure 5.4 as follows:

H_1 : For all $2^{5\times 32=160}$ possible values of $x, \Delta x, y, \Delta y$ and K_2, we obtain the corresponding values of z and Δz such that $z = y \boxminus G_{13}(x \boxplus K_2)$. If the value of Δz is equal to the difference at D_0^3, we store the values of K_2 and z in the hash table H_1 indexed by the values of $x, \Delta x, y$ and Δy. The probability that the value of Δz is equal to the difference at D_0^3 is equal to 2^{-32}. Therefore, Table H_1 has on average $2^{160} \times 2^{-32} = 2^{128}$ entries. As a result, we have, on average, $\frac{2^{128}}{2^{4\times 32}} = 1$ value for K_2 per row.

H_2 : For the value of Δx equal to the difference at D_0^3 and all 2^{24} possible value of Δy in form of the difference at C_1^3 combined with all $2^{3\times 32=96}$ possible values of x, y and K_7, we obtain the corresponding values of z and Δz such that $z = y \oplus G_{21}(x \boxplus K_7)$. Then, we store the value of K_7 in the hash table H_2 indexed by the values of x, y and Δy, if the value of Δz is equal to the difference at C_0^3 which has a probability equal to 2^{-24}. Therefore, Table H_2 has on average $2^{96+24} \times 2^{-24} = 2^{96}$ entries. Thus, we have, on average, $\frac{2^{96}}{2^{2\times 32+24}} = 2^8$ value for K_7 per row.

H_3 : For all 2^{24} possible value of Δx in form of the difference at B_1^3 combined with all $2^{4\times 32=128}$ possible values of $x, y, \Delta y$ and K_8, we obtain the corresponding values of z and Δz such that $z = y \boxplus G_{13}(x \boxplus K_8)$. If the value of Δz is in the form of the difference at A_0^3, we store the values of K_8 and z in the hash table H_3 indexed by the values of $x, \Delta x, y$ and Δy. The probability that the value of Δz is in the form of the difference at A_0^3 is equal to 2^{-32}. Therefore, Table H_3 has on average $2^{128+24} \times 2^{-32} = 2^{120}$ entries. As a result, we have, on average, $\frac{2^{120}}{2^{3\times 32+24}} = 1$ values for K_8 per row.

H_4 : Initialize a hash table of $2^{3\times 32+24=120}$ rows with binary value 0. Then, for the value of Δx equal to the difference at A_0^3 and all 2^{24} possible values of Δy in the form of the difference at B_1^3 combined with all $2^{3\times 32=96}$ possible values of x, y, and K_6, we obtain the corresponding values of z and Δz such that $z = y \oplus G_5(x \boxplus K_6)$. If the value of Δz is equal to the difference at B_0^3, we store a binary value 1 in the hash table H_4 indexed by the values of $x, y, \Delta y$ and K_6. Here, the binary values 1 and 0 denote a valid entry and an invalid entry. The probability of finding a valid entry in H_4, equivalent to the probability that the value Δz is equal to the difference at B_0^3, is equal to 2^{-24}. Consequently, we have one valid entry for every 2^{24} accesses to H_4.

Table 5.8 summarizes the time and memory complexities of the pre-computation phase. It should be noted that the memory required by the tables H_1 and H_4 can be slightly reduced to $2^{128.51}$ and $2^{119.01}$ 32-bit words respectively, if we store only the valid candidates of K_2 and K_6 based on the ratio of the valid keys form Table 5.6.

Table 5.8: The time and memory complexities of the pre-computation phase

Table	Time (S-box layer Encryption)	Memory (32-bit word)
H_1	2^{160}	$2^{160} \times 2^{-32} \times 2 = 2^{129}$
H_2	2^{120}	$2^{120} \times 2^{-24} \times 1 = 2^{96}$
H_3	2^{152}	$2^{152} \times 2^{-32} \times 2 = 2^{121}$
H_4	2^{120}	$2^{120\dagger}$

† For simplicity, we store the binary values 0 and 1 as 32-bit words.

5.5.2 Online Phase

In this phase, we collect a set of plaintext/ciphertext pairs. Then, we utilize the pre-computation tables and key guessing to obtain right candidate keys and then recover the correct master key.

Data Collection. We select a set of 2^m 128-bit plaintexts that can take any arbitrary values then we compute another set of 2^m plaintexts by XORing each plaintext in the first set with the input of the differential distinguisher (*i.e.*, $A_0^0||B_0^0||C_0^0||D_0^0$). After that, we query the encryption oracle and compute the corresponding ciphertext difference. Here, we use 2^{m+1} plaintexts to generate 2^m plaintext/ciphertext pairs satisfying the input difference of our differential distinguisher (the value of m will be determined below).

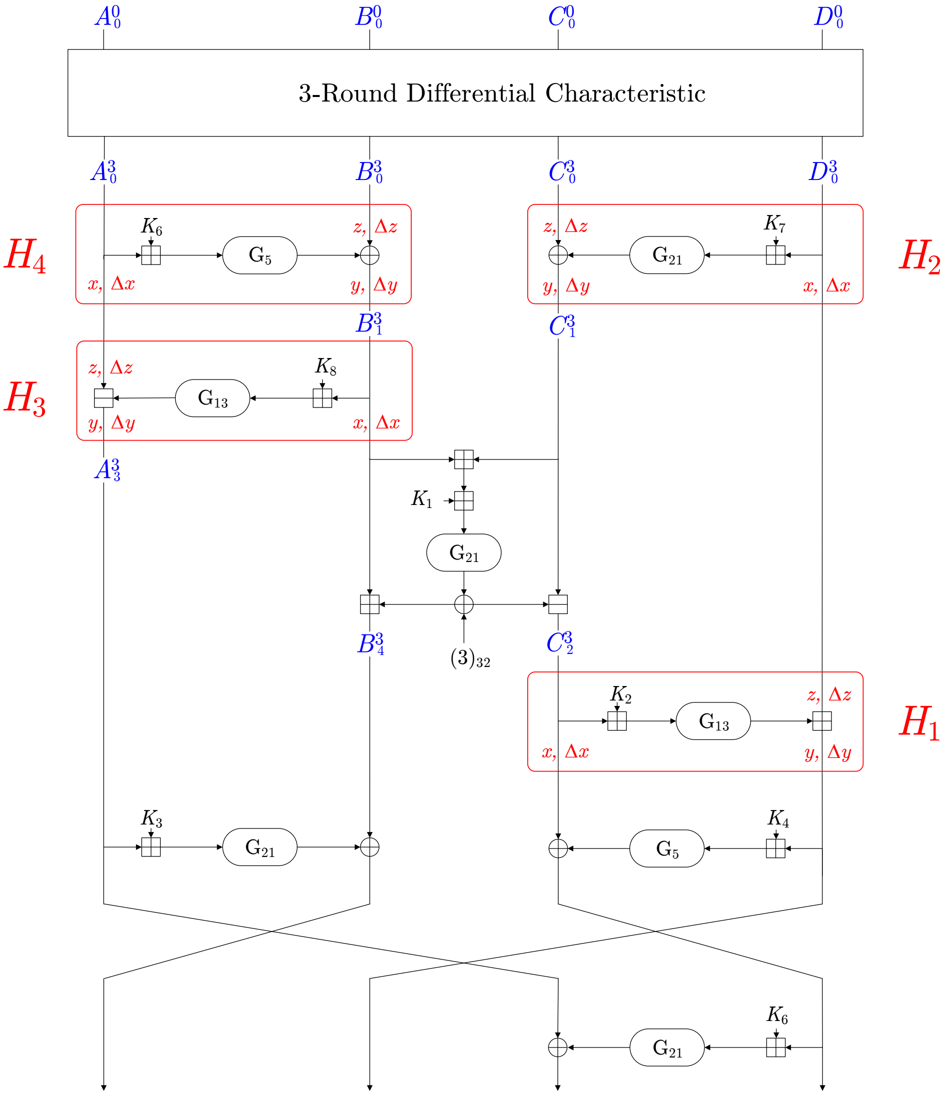

Figure 5.4: $4\frac{1}{7}$-round attack on Bel-T-256

Key Recovery. We first prepare $2^{7\times 32} = 2^{224}$ counters corresponding to the 2^{224} keys involved in the analysis. After that, for each ciphertext pair in 2^m pairs obtained in the data collection phase, we apply the following procedure:

1. Guess K_4 and partially decrypt the ciphertext to get the value and the difference at C_2^3. The average number of keys suggested by a pair after this step is 2^{32}.

2. Access the hash table H_1 to get, on average, 1 value of K_2 and D_0^3.

3. Guess K_6 and partially decrypt the ciphertext to get the value and the difference at A_3^3. The average number of keys suggested by a pair after this step will increase to 2^{64}.

4. Guess K_3 and partially decrypt the ciphertext combined with the value and the difference from the previous step to get the value and the difference at B_4^3. The average number of keys suggested by a pair after this step is 2^{96}.

5. Recall that $B_1^3 = B_4^3 \boxminus G_{21}(B_1^3 \boxplus C_1^3 \boxplus K_1) \oplus (3)_{32}$ and $C_1^3 = C_2^3 \boxplus G_{21}(B_1^3 \boxplus C_1^3 \boxplus K_1) \oplus (3)_{32}$. Hence $B_1^3 \boxplus C_1^3 = B_4^3 \boxplus C_2^3$. Therefore, by guessing K_1, we can deduce $G_{21}(B_1^3 \boxplus C_1^3 \boxplus K_1) = G_{21}(B_4^3 \boxplus C_2^3 \boxplus K_1)$ and then use the values obtained in steps 1 and 4 to compute the value and the difference at B_1^3 and C_1^3 and discard the key if the differences are not in the required form. This step filters out the suggested keys by 2^{16}. Thus, the average number of keys suggested by a pair after this step is 2^{112}.

6. Use the values and the differences form steps 3 and 5 to access the hash table H_3 and get, on average, 1 values of K_8 and A_0^3.

7. Access the hash table H_4 using the previously guessed value of K_6 in step 3 and the values and the differences from steps 5 and 7 to check if it is a valid entry or not. This step will filter out the suggested keys by 2^{24}. Thus, the average number of keys suggested by a pair after this filtration will be 2^{88}.

Table 5.9: Key recovery process of the attack on $4\frac{1}{7}$-round Bel-T-256

Step	# of suggested keys by a pair	Time Complexity	
		32-bit word memory Access	S-box layer Encryption
1	2^{32}	-	$2^m \times 2^{32} \times 2 = 2^{m+33}$
2	$2^{32} \times 1 = 2^{32}$	$2^m \times 2^{32} \times 2 = 2^{m+33}$	-
3	$2^{32} \times 2^{32} = 2^{64}$	-	$2^m \times 2^{64} \times 2 = 2^{m+65}$
4	$2^{64} \times 2^{32} = 2^{96}$	-	$2^m \times 2^{96} \times 2 = 2^{m+97}$
5	$2^{96} \times 2^{32} \times 2^{-16} = 2^{112}$	-	$2^m \times 2^{128} \times 2 = 2^{m+129}$
6	$2^{112} \times 1 = 2^{112}$	$2^m \times 2^{112} \times 2 = 2^{m+113}$	-
7	$2^{112} \times 2^{-24} = 2^{88}$	$2^m \times 2^{112} \times 1 = 2^{m+112}$	-
8	$2^{88} \times 2^{8} = 2^{96}$	$2^m \times 2^{96} \times 1 = 2^{m+96}$	-

8. Use the value from step 2 combined with the value and the difference from step 5 to access the hash table H_2 and get, on average, 2^8 value of K_7. Consequently, the average number of keys suggested by a pair after this procedure will be increased to 2^{96}. Thus, we increment the corresponding 2^{96} counters.

After repeating the above procedure for 2^m pairs, we select the key corresponding to the highest counter as a 224-bit right key. After that, we recover the 256-bit master key by testing the 224-bit right key along with the remaining 2^{32} values for K_5 using 2 plaintext/ciphertext pairs.

Table 5.9 summarizes the above steps, whereas the second column presents the average number of keys suggested by a pair after each step. The third and fourth columns present the time complexity of each step in form of memory accesses and single S-box layer encryption in terms of m.

5.5.3 Attack Complexity and Success Probability

In this section, we present the complexity analysis of our attack in order to determine the required number of chosen plaintexts and the memory required to launch this attack. Also, we compute the success probability of the attack. Finally, we calculate its time complexity to compare our attack against the exhaustive search attack.

Data Complexity. For the differential attack to succeed with a high probability, we have to determine an appropriate value for the number of required plaintext/ciphertext pairs. To do so, we utilize the concept of signal-to-noise ratio (S/N) [13], which is calculated using the following formula:

$$S/N = \frac{2^k \times p}{\alpha \times \beta}$$

where k is the number of key bits involved in the analysis, p is the probability of the differential characteristic, α is the number of guessed keys by a pair, and β is the ratio of the pairs that are not discarded. In our analysis, $k = 224$, $p = 2^{-111}$, $\alpha = 2^{96}$ from table 5.9, and $\beta = 1$. Therefore, we have $S/N = \frac{2^{224} \times 2^{-111}}{2^{96} \times 1} = 2^{17}$. Due to this high S/N, we can use the recommendation of Biham and Shamir [13] that $3 \sim 4$ right pairs are sufficient enough to mount a successful differential attack. Therefore, we select the number of plaintext/ciphertext pairs (2^m) equal to $4 \times p^{-1} = 2^{113}$. Consequently, the data complexity will be 2^{114} chosen plaintexts.

According to [84] and due to the high S/N, the success probability of the attack (P_s) can be calculated as $P_s \approx \Phi(\sqrt{p \times 2^m})$ where Φ is the cumulative distribution function of the standard normal distribution. Therefore, our differential attack will succeed with probability $P_s \approx 0.9772$.

Time Complexity. During the attack procedure, we make 32-bit word memory accesses in some steps and partially decrypt single S-box layers in other steps. Each S-box layer can be considered as a 32-bit big S-box with one or two modulo operations. Therefore, the time of single S-box layer will be slightly higher than the time of 32-bit word memory access. For simplicity, we assume that the time of 32-bit word memory access is the same as the time of a single S-box layer lookup which is roughly equal to $\frac{1}{7}$ of the time of one round encryption. From Table 5.8, the time complexity of the pre-computation phase is dominated by the time required to construct the hash table H_1 which is equal to $\frac{1}{7} \times \frac{1}{4\frac{1}{7}} \times 2^{160} \approx 2^{155.14}$ $4\frac{1}{7}$-round encryptions. Similarly, from Table 5.9, the dominant part of the time complexity in the online phase comes from steps 5 which is $\frac{1}{7} \times \frac{1}{4\frac{1}{7}} \times (2^{m+129}) = 2^{m+124.14}$ $4\frac{1}{7}$-round encryptions. Therefore, the total time complexity of the online phase will be $2^{113+124.14} + 2 \times 2^{32} = 2^{237.14}$ $4\frac{1}{7}$-round encryptions.

Memory Complexity. The memory complexity of the pre-computation phase can be determined from Table 5.8 in which we need $2^{129} + 2^{96} + 2^{121} + 2^{120} \approx 2^{129}$ 32-bit word $=$ 2^{127} 128-bit blocks of memory. During the online phase, we have prepared 2^{224} counters corresponding to 2^{224} keys involved in the analysis. Since the upper limit of each counter depends on the number of plaintext/ciphertext pairs ($2^m = 2^{113}$), we can declare each counter as an unsigned 128-bit integer variable. Consequently, we need 2^{224} 128-bit blocks of memory in total.

5.6 Summary

In this chapter, we studied the propagation of the XOR difference through modular addition. We showed that the independency assumption between two or more consecutive modular addition operations does not necessarily hold, and we constructed a more accurate MILP model for the differential trail through the modular addition taking into account the dependency between the consecutive modular additions. Then, we utilized the developed MILP model to automate the search process for the differential characteristics for Bel-T cipher. Up to the authors' knowledge, this is the best published theoretical attack against Bel-T-256 in the single-key setting.

Chapter 6

Integral Attacks on Round-Reduced Bel-T-256

In this chapter, we continue investigating the security of Bel-T-256 [10]. Precisely, we present integral attacks against Bel-T-256 using the propagation of the bit-based division property. Firstly, we propose two 2-round integral characteristics by employing a Mixed Integer Linear Programming (MILP) approach to propagate the division property through the round function. Then, we utilize these integral characteristics to attack $3\frac{2}{7}$ rounds (out of 8) Bel-T-256 with data and time complexities of 2^{13} chosen plaintexts and $2^{199.33}$ encryption operations, respectively. We also present an attack against $3\frac{6}{7}$ rounds with data and time complexities of 2^{33} chosen plaintexts and $2^{254.61}$ encryption operations, respectively. To the best of our knowledge, these attacks are the first published theoretical attacks against the cipher in the single-key setting.

6.1 Introduction

As mentioned in the previous chapter, the Republic of Belarus has approved the Bel-T block cipher family as the state standard cryptographic encryption algorithm in 2011 [10]. The Bel-T family consists of three block ciphers, denoted as Bel-T-k, with the

Table 6.1: Attack results on Bel-T-256

Model	Attack	#Round	Data	Time	Reference
Related Key	Differential	$5\frac{6}{7}$	$2^{123.28}$	$2^{228.4}$	[2]
Single Key	Integral	$3\frac{2}{7}$	2^{13}	$2^{199.33}$	Sec. 6.3.2
		$3\frac{6}{7}$	2^{33}	$2^{254.61}$	Sec. 6.3.3

same block size of 128 bits and key length k = 128, 192 or 256 bits. Bel-T merges a Lai-Massey scheme [62] with a Feistel network [39]. At the time of publishing this work in [34], there were only two published cryptanalysis results on Bel-T's; fault-based attacks was considered in [53], and related-key differential attack on round-reduced Bel-T-256 was presented in [2]. In this chapter, we present the first published single-key attack against Bel-T-256. Table 6.1 contrasts the result of our attacks with the related-key differential attack in [2].

The rest of this chapter is organized as follows. In Section 6.2, we briefly revisit the bit-based division property and summarize how to present its propagation through the basic cipher operations with MILP models. We also describe our approach to model the modular subtraction operation. In Section 6.3, we investigate the security of Bel-T block cipher against the integral attacks utilizing the MILP approach. Finally, the summary is presented in Section 6.4.

6.2 Bit-based Division Property

As mentioned in Chapter 2, the division property [95] is a generalization of the integral property to utilize the hidden relations between the traditional $\mathcal{A}$ and $\mathcal{B}$ properties by exploiting the algebraic degree of the nonlinear components of the block cipher. The division property was succeeded by a more precise version called the *bit-based division property* (BDP) in [98] which exploits the internal structure of the nonlinear components to analyze block ciphers at the bit level.

6.2.1 MILP Models for the Bit-based Division Property

As mentioned in Chapter 2, with the help of the division trail, it becomes easy to employ MILP for constructing the integral distinguisher. In the following, we briefly describe how to model the division trail through several operations as MILP constraints.

Model for COPY [90] Let $(a) \xrightarrow{COPY} (b_1, b_2, \ldots, b_m)$ denote the division trail through COPY function, where one bit is copied to m bits. Then, it can be described using the following MILP constraints:

$$a - b_1 - b_2 - \cdots - b_m = 0, \text{ where } a, b_1, b_2, \ldots, b_m \text{ are binary variables.}$$

Model for XOR [90] Let $(a_1, a_2, \ldots, a_m) \xrightarrow{XOR} (b)$ denote the division trail through an XOR function, where m bits are compressed to one bit using an XOR operation. Then, it can be described using the following MILP constraints:

$$a_1 + a_2 + \cdots + a_m - b = 0, \text{ where } a_1, a_2, \ldots, a_m, b \text{ are binary variables.}$$

Model for AND [104] Let $(a_0, a_1) \xrightarrow{AND} (b)$ denote the division trail though an AND function, where two bits are compressed using an AND operation. Then, it can be described using the following MILP constraints:

$$b - a_0 \geq 0, \quad b - a_1 \geq 0, \quad \text{where } a_0, a_1, b \text{ are binary variables.}$$

MILP Model for S-boxes. The bit-based division property introduced in [98] is limited to bit-orientated ciphers and cannot be applied to ciphers with S-boxes. Xiang *et al.* [104] complemented this work by proposing an algorithm to accurately compute the bit-based division property through an S-box. Briefly, they represented the S-box using its algebraic normal form (ANF). Then,the division trail though an n-bit S-box can be represented as a set of $2n$-dimensional binary vectors $\in \{0, 1\}^{2n}$ which has a convex hull. The H-

Representation of this convex hull can be computed using readily available functions such as `inequality_generator()` function in SageMath[*] which returns a set of linear inequalities that describe these vectors. We use this set of inequalities as MILP constraints to present the division trail though the S-box.

MILP Model for Modular Addition. In [88], Sun *et al.* proposed a systematic method to deduce an MILP model for the modular addition operation of 4-bit variables by expressing the operation at the bit-level. Then this method is generalized for n-bit variables in [89]. Let $\boldsymbol{x} = (x_0, x_1, \ldots, x_{n-1})$, $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1})$, and $\boldsymbol{z} = (z_0, z_1, \ldots, z_{n-1})$[†] be n-bit vectors where $\boldsymbol{z} = \boldsymbol{x} \boxplus \boldsymbol{y}$. Then, z_i can be iteratively expressed as follows:

$$z_{n-1} = x_{n-1} \oplus y_{n-1} \oplus c_{n-1},\ c_{n-1} = 0,$$

$$z_i = x_i \oplus y_i \oplus c_i,\ c_i = x_{i+1}y_{i+1} \oplus (x_{i+1} \oplus y_{i+1})c_{i+1}, \quad i = n-2, n-3, \ldots, 0.$$

Consequently, the division trail through the modular addition can be deduced in terms of COPY, AND, and XOR operations [89].

MILP Model for Modular Addition with a Constant. In [88], Sun *et al.* explain how to deduce an MILP model for the modular addition of a 4-bit variable with a constant. The authors expressed the operation at the bit-level and exploited that the operations of XOR/AND with a constant do not influence the division property [88]. We can generalize this method for n-bit variables as follows. Let $(a_0, a_1, \ldots, a_{n-1}) \rightarrow (d_0, d_1, \ldots, d_{n-1})$ denote the division trail through n-bit modular addition with a constant, the division property propagation can be decomposed as COPY, AND, and XOR operations as follows:

[*] http://www.sagemath.org/

[†] big-endian representation

$$\begin{cases}
(a_{n-1}) \xrightarrow{COPY} (d_{n-1}, f_0, g_0) \\
(a_{n-2}) \xrightarrow{COPY} (a_{n-2,0}, a_{n-2,1}, a_{n-2,2}) \\
(a_{n-2,0}, f_0) \xrightarrow{XOR} (d_{n-2}) \\
(a_{n-2,1}, g_0) \xrightarrow{AND} (e_0) \\
(a_{n-2,2}, e_0) \xrightarrow{XOR} (v_0) \\
\left.\begin{array}{l}
(v_{i-1}) \xrightarrow{COPY} (f_i, g_i) \\
(a_{n-2-i}) \xrightarrow{COPY} (a_{n-2-i,0}, a_{n-2-i,1}, a_{n-2-i,2}) \\
(a_{n-2-i,0}, f_i) \xrightarrow{XOR} (d_{n-2-i}) \\
(a_{n-2-i,1}, g_i) \xrightarrow{AND} (e_i) \\
(a_{n-2-i,2}, e_i) \xrightarrow{XOR} (v_i)
\end{array}\right\} iterated\ for\ i = 1, ..., n-3 \\
(a_0, v_{n-3}) \xrightarrow{XOR} (d_0)
\end{cases}$$

where the intermediate variables $a_{i,0}$, $a_{i,1}$, $a_{i,2}$, f_i, g_i, e_i, and v_i are as shown in Table 6.2.

MILP Model for Modular Subtraction. In this section, we present an approach to deduce an MILP model for the modular subtraction operation using the same methodology used for Modular Addition. For consistency, we use the same notation as in [88].

Let $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ be n-bit vectors where $\boldsymbol{z} = \boldsymbol{x} \boxminus \boldsymbol{y}$. This relation can be rewritten as $\boldsymbol{z} = \boldsymbol{x} \boxplus$ (2's complement of $\boldsymbol{y}$) $= \boldsymbol{x} \boxplus (\bar{\boldsymbol{y}} \boxplus 1)$, where $\bar{\boldsymbol{y}}$ is the 1's complement of $\boldsymbol{y}$. Therefore, the division trail through the modular subtraction can be modelled as a division trail through a modular addition followed by a modular addition with a constant. This representation has two issues. The first issue is that two operations are used to present one operation which requires the use of more MILP constraints and variables, and consequently slowing down the search process. The second issue is that the information

Table 6.2: The intermediate variables for modular addition with a constant

$\underbrace{z_{n-1}}_{d_{n-1}}$	$\underbrace{x_{n-1}}_{a_{n-1}}$		
$\underbrace{z_{n-2}}_{d_{n-2}}$	$\underbrace{x_{n-2}}_{a_{n-2,0}} \oplus \underbrace{c_{n-2}}_{f_0}$	c_{n-2}	x_{n-1}
$\underbrace{z_{n-3}}_{d_{n-3}}$	$\underbrace{x_{n-3}}_{a_{n-3,0}} \oplus \underbrace{c_{n-3}}_{f_1}$	$\underbrace{c_{n-3}}_{v_0}$	$\underbrace{x_{n-2}}_{a_{n-2,2}} \oplus \overbrace{\underbrace{x_{n-2}}_{a_{n-2,1}} \underbrace{c_{n-2}}_{g_0}}^{e_0}$
$\underbrace{z_{n-4}}_{d_{n-4}}$	$\underbrace{x_{n-4}}_{a_{n-4,0}} \oplus \underbrace{c_{n-4}}_{f_2}$	$\underbrace{c_{n-4}}_{v_1}$	$\underbrace{x_{n-3}}_{a_{n-3,2}} \oplus \overbrace{\underbrace{x_{n-3}}_{a_{n-3,1}} \underbrace{c_{n-3}}_{g_1}}^{e_1}$
	$\dots$	$\dots$	
$\underbrace{z_1}_{d_1}$	$\underbrace{x_1}_{a_{1,1}} \oplus \underbrace{c_1}_{f_{n-3}}$	$\underbrace{c_1}_{v_{n-4}}$	$\underbrace{x_2}_{a_{2,2}} \oplus \overbrace{\underbrace{x_2}_{a_{2,1}} \underbrace{c_2}_{g_{n-4}}}^{e_{n-4}}$
$\underbrace{z_0}_{d_0}$	$\underbrace{x_0}_{a_0} \oplus c_0$	$\underbrace{c_0}_{v_{n-3}}$	$\underbrace{x_1}_{a_{1,2}} \oplus \overbrace{\underbrace{x_1}_{a_{1,1}} \underbrace{c_1}_{g_{n-3}}}^{e_{n-3}}$

about the value of the constant, which is 1, in the modular addition with a constant is not utilized. This may lead the search process to conclude that some bits are not balanced even that they are balanced, as we show in the following section (Section 6.2.2). Instead, at the bit level implementation, the modular subtraction operation is handled as a modular addition operation with two modifications: the first carry to the modular addition will be 1 instead of 0 ($c_{n-1} = 1$), and the second input to the modular addition will be the 1's complement of the second operand ($\bar{\boldsymbol{y}}$).

Let $\boldsymbol{x} = (x_0, x_1, \dots, x_{n-1})$, $\boldsymbol{y} = (y_0, y_1, \dots, y_{n-1})$, and $\boldsymbol{z} = (z_0, z_1, \dots, z_{n-1})$. Then, z_i can be iteratively expressed as follows:

$$z_{n-1} = x_{n-1} \oplus \bar{y}_{n-1} \oplus c_{n-1},\ c_{n-1} = 1,$$

$$z_i = x_i \oplus \bar{y}_i \oplus c_i,\ c_i = x_{i+1}\bar{y}_{i+1} \oplus (x_{i+1} \oplus \bar{y}_{i+1})c_{i+1}, \quad \forall i = n-2, n-3, \dots, 0.$$

$$where\ \bar{y}_i = y_i \oplus 1$$

The operation of XOR/AND with a constant does not influence the division property [88]. Therefore, the division property of $\bar{\boldsymbol{y}}$ is the same of $\boldsymbol{y}$.

Table 6.3: The intermediate variables for modular subtraction

$\underbrace{z_{n-1}}_{d_{n-1}} = \underbrace{x_{n-1}}_{a_{n-1,0}} \oplus \underbrace{\bar{y}_{n-1}}_{b_{n-1,0}} \oplus 1$	
$\underbrace{z_{n-2}}_{d_{n-2}} = \underbrace{x_{n-2}}_{a_{n-2,0}} \oplus \underbrace{\bar{y}_{n-2}}_{b_{n-2,0}} \oplus \overbrace{c_{n-2}}^{g_0}$	$\overbrace{c_{n-2}}^{v_0} = \overbrace{\underbrace{x_{n-1}}_{a_{n-1,1}} \underbrace{\bar{y}_{n-1}}_{b_{n-1,1}}}^{t_1} \oplus (\overbrace{\underbrace{x_{n-1}}_{a_{n-1,2}} \oplus \underbrace{\bar{y}_{n-1}}_{b_{n-1,2}}}^{t_0})$
$\underbrace{z_{n-3}}_{d_{n-3}} = \underbrace{x_{n-3}}_{a_{n-3,0}} \oplus \underbrace{\bar{y}_{n-3}}_{b_{n-3,0}} \oplus \overbrace{c_{n-3}}^{g_1}$	$\overbrace{c_{n-3}}^{w_0} = \overbrace{\underbrace{x_{n-2}}_{a_{n-2,1}} \underbrace{\bar{y}_{n-2}}_{b_{n-2,1}}}^{v_1} \oplus \overbrace{(\overbrace{\underbrace{x_{n-2}}_{a_{n-2,2}} \oplus \underbrace{\bar{y}_{n-2}}_{b_{n-2,2}}}^{m_0}) \oplus \overbrace{c_{n-2}}^{r_0}}^{q_0}$
$\underbrace{z_{n-4}}_{d_{n-4}} = \underbrace{x_{n-4}}_{a_{n-4,0}} \oplus \underbrace{\bar{y}_{n-4}}_{b_{n-4,0}} \oplus \overbrace{c_{n-4}}^{g_2}$	$\overbrace{c_{n-4}}^{w_1} = \overbrace{\underbrace{x_{n-3}}_{a_{n-3,1}} \underbrace{\bar{y}_{n-3}}_{b_{n-3,1}}}^{v_2} \oplus \overbrace{(\overbrace{\underbrace{x_{n-3}}_{a_{n-3,2}} \oplus \underbrace{\bar{y}_{n-3}}_{b_{n-3,2}}}^{m_1}) \oplus \overbrace{c_{n-3}}^{r_1}}^{q_1}$
$\cdots$	$\cdots$
$\underbrace{z_1}_{d_1} = \underbrace{x_1}_{a_{1,0}} \oplus \underbrace{\bar{y}_1}_{b_{1,0}} \oplus \overbrace{c_1}^{g_{n-3}}$	$\overbrace{c_1}^{w_{n-4}} = \overbrace{\underbrace{x_2}_{a_{2,1}} \underbrace{\bar{y}_2}_{b_{2,1}}}^{v_{n-3}} \oplus \overbrace{(\overbrace{\underbrace{x_2}_{a_{2,2}} \oplus \underbrace{\bar{y}_2}_{b_{2,2}}}^{m_{m-4}}) \oplus \overbrace{c_2}^{r_{n-4}}}^{q_{n-4}}$
$\underbrace{z_0}_{d_0} = \underbrace{x_0}_{a_0} \oplus \underbrace{\bar{y}_0}_{b_0} \oplus c_0$	$\overbrace{c_0}^{w_{n-3}} = \overbrace{\underbrace{x_1}_{a_{1,1}} \underbrace{\bar{y}_1}_{b_{1,1}}}^{v_{n-2}} \oplus \overbrace{(\overbrace{\underbrace{x_1}_{a_{1,2}} \oplus \underbrace{\bar{y}_1}_{b_{1,2}}}^{m_{m-3}}) \oplus \overbrace{c_1}^{r_{n-3}}}^{q_{n-3}}$

Consequently, we can generalize the modular subtraction operation for n-bit variables as follows:

Proposition 6.1 *Let $((a_0, a_1, \ldots, a_{n-1}), (b_0, b_1, \ldots, b_{n-1})) \to (d_0, d_1, \ldots, d_{n-1})$ be a division trail through n-bit modular subtraction operation. The division property propagation can be decomposed as COPY, AND, and XOR operations as follows where the intermediate variables $a_{i,0}$, $a_{i,1}$, $a_{i,2}$, t_0, t_1, v_i, g_i, r_i, m_i, q_i, and w_i are as shown in Table 6.3.*

$$\begin{cases}
(a_{n-1}) \xrightarrow{COPY} (a_{n-1,0}, a_{n-1,1}, a_{n-1,2}) \\
(b_{n-1}) \xrightarrow{COPY} (b_{n-1,0}, b_{n-1,1}, b_{n-1,2}) \\
(a_{n-1,0}, b_{n-1,0}) \xrightarrow{XOR} (d_{n-1}) \\
(a_{n-1,2}, b_{n-1,2}) \xrightarrow{XOR} (t_0) \\
(a_{n-1,1}, b_{n-1,1}) \xrightarrow{AND} (t_1) \\
(t_0, t_1) \xrightarrow{XOR} (v_0) \\
(v_0) \xrightarrow{COPY} (g_0, r_0) \\
(a_{n-2}) \xrightarrow{COPY} (a_{n-2,0}, a_{n-2,1}, a_{n-2,2}) \\
(b_{n-2}) \xrightarrow{COPY} (b_{n-2,0}, b_{n-2,1}, b_{n-2,2}) \\
\left.\begin{array}{l}
(a_{n-i,0}, b_{n-i,0}, g_{i-2}) \xrightarrow{XOR} (d_{n-i}) \\
(a_{n-i,1}, b_{n-i,1}) \xrightarrow{AND} (v_{i-1}) \\
(a_{n-i,2}, b_{n-i,2}) \xrightarrow{XOR} (m_{i-2}) \\
(m_{i-2}, r_{i-2}) \xrightarrow{AND} (q_{i-2}) \\
(v_{i-1}, q_{i-2}) \xrightarrow{XOR} (w_{i-2}) \\
(w_{i-2}) \xrightarrow{COPY} (g_{i-1}, r_{i-1}) \\
(a_{n-i-1}) \xrightarrow{COPY} (a_{n-i-1,0}, a_{n-i-1,1}, a_{n-i-1,2}) \\
(b_{n-i-1}) \xrightarrow{COPY} (b_{n-i-1,0}, b_{n-i-1,1}, b_{n-i-1,2})
\end{array}\right\} iterated\ for\ i = 2, ..., n-2 \\
(a_{1,0}, b_{1,0}, g_{n-3}) \xrightarrow{XOR} (d_1) \\
(a_{1,1}, b_{1,1}) \xrightarrow{AND} (v_{n-2}) \\
(a_{1,2}, b_{1,2}) \xrightarrow{XOR} (m_{n-3}) \\
(m_{n-3}, r_{n-3}) \xrightarrow{AND} (q_{n-3}) \\
(v_{n-2}, q_{n-3}) \xrightarrow{XOR} (w_{n-3}) \\
(a_0, b_0, w_{n-3}) \xrightarrow{XOR} (d_0)
\end{cases}$$

6.2.2 Validity of our MILP Model for Modular Subtraction

In this section, we provide the result of our experiments on a toy cipher in order to validate the MILP model for the division trail through a modular subtraction operation. Moreover, we show that the proposed model of the division trail through the modular subtraction at the bit-level ($\boldsymbol{z} = \boldsymbol{x} \boxminus \boldsymbol{y}$) gives better results than modelling it as a division trail through a modular addition followed by a modular addition with a constant (*i.e.*, $\boldsymbol{z} = \boldsymbol{x} \boxplus \bar{\boldsymbol{y}} \boxplus 1$).

The round function of the toy cipher used during the experiments is a small version of the SPECK round function [5] with modular subtraction instead of modular addition as shown in Figure 6.1 where the block size is 8 bits, (X_L^i, X_R^i) is the input of the i-th round, and k_i is the subkey used in the i-th round.

We follow the same approach used in [88] to validate their MILP model for modular addition. The experimental procedure is as follows:

1. For an initial division property, use our MILP model for the modular subtraction at the bit-level ($\boldsymbol{z} = \boldsymbol{x} \boxminus \boldsymbol{y}$) to find the set of balanced bits at the output of the toy cipher.

2. Use the other MILP model ($\boldsymbol{z} = \boldsymbol{x} \boxplus \bar{\boldsymbol{y}} \boxplus 1$) to find the balanced bits corresponding the same initial division property.

3. Exhaustively search for the balanced bits as follows:

   (a) Divide the space of the plaintexts (2^8 plaintexts) to a group of multi-sets of plaintexts. Each one of these multi-sets satisfies the initial division property.

   (b) Encrypt each multi-set of the plaintexts using a randomly chosen key and find the bits with zero-sum over all the corresponding ciphertexts of that multi-set, and then find the common zero-sum bits over all the multi-sets.

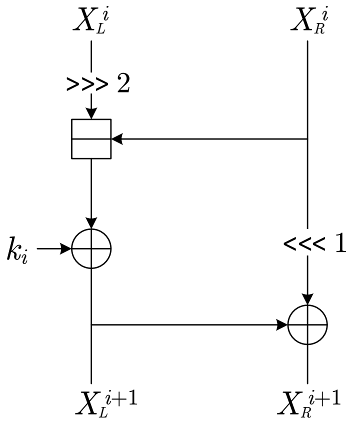

Figure 6.1: The round function of the toy cipher.

(c) Repeat the previous step 2^{10} iterations and find the common zero-sum bits at the output of the toy cipher over all the iterations.

4. Compare the results from the previous three steps for the same initial division property.

5. Repeat the previous steps for all possible values of the initial division property and for a toy cipher consists of up to 6 rounds similar to the one in the Figure 6.1.

From the result of the experiments, we can conclude that the balanced bits found by the MILP-aided bit-based division property are indeed balanced. Moreover, the MILP model for the division trail through the modular subtraction at the bit-level ($\boldsymbol{z} = \boldsymbol{x} \boxminus \boldsymbol{y}$) also uses less number of constraints and gives same or better results (in terms of number of the balanced bits) than modelling it as a division trail through a modular addition followed by a modular addition with a constant ($\boldsymbol{z} = \boldsymbol{x} \boxplus \bar{\boldsymbol{y}} \boxplus 1$). A sample of our results can be found in Table 6.4 and the mismatch between the two approaches for modelling the division trail through a modular subtractions is summarized in Table 6.5.

Table 6.4: Comparison of zero-sum bits found by using three methods for the toy cipher, where #{Bits} is the number of balanced bits and 'Bits' is the position of these bits counted from the most significant bit.

Input Division property	Rounds	Exhaustive search		MILP-aided Bit-based Division property			
				$z = x \boxminus y$		$z = x \boxplus \bar{y} \boxplus 1$	
		#{Bits}	Bits	#{Bits}	Bits	#{Bits}	Bits
$\mathcal{D}^{1^8}_{\{[01111111]\}}$	1	8	$0 \sim 7$	8	$0 \sim 7$	8	$0 \sim 7$
	2	8	$0 \sim 7$	8	$0 \sim 7$	8	$0 \sim 7$
	3	6	$1 \sim 3,5 \sim 7$	6	$1 \sim 3,5 \sim 7$	4	$2 \sim 3,6 \sim 7$
	4	1	3	1	3	0	-
	5	0	-	0	-	0	-
$\mathcal{D}^{1^8}_{\{[11111110]\}}$	1	8	$0 \sim 7$	8	$0 \sim 7$	8	$0 \sim 7$
	2	8	$0 \sim 7$	8	$0 \sim 7$	8	$0 \sim 7$
	3	6	$1 \sim 3,5 \sim 7$	6	$1 \sim 3,5 \sim 7$	6	$1 \sim 3,5 \sim 7$
	4	3	$2 \sim 3, 6$	1	3	1	3
	5	0	-	0	-	0	-
$\mathcal{D}^{1^8}_{\{[00001111]\}}$	1	8	$0 \sim 7$	8	$0 \sim 7$	8	$0 \sim 7$
	2	4	$2 \sim 3,6 \sim 7$	4	$2 \sim 3,6 \sim 7$	4	$2 \sim 3,6 \sim 7$
	3	0	-	0	-	0	-
$\mathcal{D}^{1^8}_{\{[11110000]\}}$	1	8	$0 \sim 7$	8	$0 \sim 7$	8	$0 \sim 7$
	2	8	$0 \sim 7$	8	$0 \sim 7$	6	$1 \sim 3, 5 \sim 7$
	3	2	$3, 7$	2	$3, 7$	1	3
	4	0	-	0	-	0	-

6.3 Integral Attack on Bel-T-256

In this Section, we investigate the security of the Bel-T block cipher against the integral attack based on the bit-based division property. For Bel-T specification, we refer the reader to Section 5.4.1 in the previous chapter.

6.3.1 Integral Distinguishers of Bel-T

The Bel-T round function includes 7 S-boxes, modular additions, modular additions with key and modular subtractions. We construct an MILP model for the bit-based division property through Bel-T as follows. Firstly, we generate the division trail of the S-box using Algorithm 2 in [104]. Then, we deduce the inequalities of the S-box using `inequality_generator()` function in Sage. In the case of the Bel-T's S-box, the number of generated inequalities is 71736, which is very large set to be handled by any MILP

Table 6.5: Mismatch between the two approaches for modelling the division trail through a modular subtraction.

Rounds	Inputs Division property	MILP-aided Bit-based Division property			
		$z = x \boxminus y$		$z = x \boxplus \bar{y} \boxplus 1$	
		#{Bits}	Bits	#{Bits}	Bits
1	{[10000011]}, {[11000010]}, {[11000011]}	8	$0 \sim 7$	6	$1 \sim 3, 5 \sim 7$
2	{[01101101]}, {[01111001]}, {[10100101]}, {[10101100]}, {[10110001]}, {[10111000]}, {[11100100]}, {[11110000]}	8	$0 \sim 7$	6	$1 \sim 3, 5 \sim 7$
	{[10001111]}, {[10011011]}, {[11001110]}, {[11001111]}, {[11011010]}, {[11011011]}	6	$1 \sim 3, 5 \sim 7$	4	$2 \sim 3, 6 \sim 7$
	{[10000011]}, {[11000010]}	2	3,7	1	3
	{[11000011]}	4	$2 \sim 3, 6 \sim 7$	1	3
3	{[01110111]}, {[01111111]}, {[10110110]}, {[10110111]}, {[10111101]}, {[10111110]}, {[11110110]}, {[11111100]}	6	$1 \sim 3, 5 \sim 7$	4	$2 \sim 3, 6 \sim 7$
	{[01101101]}, {[01111001]}, {[10100101]}, {[10101100]}, {[10110001]}, {[10111000]}, {[11100100]}, {[11110000]}	2	3,7	1	3
	{[10001111]}, {[10011011]}, {[11001110]}, {[11001111]}, {[11011010]}, {[11011011]}	1	3	0	-
4	{[11111011]}	6	$1 \sim 3, 5 \sim 7$	4	$2 \sim 3, 6 \sim 7$
	{[01110111]}, {[01111111]}, {[10110110]}, {[10110111]}, {[10111101]}, {[10111110]}, {[11110110]}, {[11111100]}	1	3	0	-
5	{[11111011]}	1	3	0	-

optimizer. Therefore, we reduce this set using a Greedy Algorithm which is proposed by Sun *et al.* in [91]. As a result, the set of inequalities represented the Bel-T's S-box is reduced to 28.

Then, we implement the MILP model for modular addition and deduce the model for subtraction. Finally, we use the Gurobi optimizer [48] to search for the longest integral distinguisher for Bel-T. Based on our implementation, we found several 2-round integral distinguishers. Our code that is used to generate the MILP model for Bel-T and to search for an integral distinguisher can be downloaded from github.[‡]

[‡] https://github.com/mhgharieb/Bel-T-256

In here, we present two such distinguishers which are chosen in order to minimize the attack data and time complexities.

$$
\begin{aligned}
IC1: \quad & ((\mathcal{C}_{0-31}), (\mathcal{C}_{0-31}), (\mathcal{C}_{0-17}||\mathcal{A}_{18-18}||\mathcal{C}_{19-31}), (\mathcal{A}_{0-7}||\mathcal{C}_{8-31})) \\
& \xrightarrow{2R} ((\mathcal{U}_{0-31}), (\mathcal{U}_{0-31}), (\mathcal{U}_{0-26}||\mathcal{B}_{27-31}), (\mathcal{U}_{0-31})) \\
IC2: \quad & ((\mathcal{C}_{0-31}), (\mathcal{C}_{0-31}), (\mathcal{C}_{0-10}||\mathcal{A}_{11-26}||\mathcal{C}_{27-31}), (\mathcal{A}_{0-15}||\mathcal{C}_{16-31})) \\
& \xrightarrow{2R} ((\mathcal{U}_{0-26}||\mathcal{B}_{27-31}), (\mathcal{U}_{0-31}), (\mathcal{B}_{0-31}), (\mathcal{U}_{0-31}))
\end{aligned}
$$

where $\mathcal{C}_{i-j}/\mathcal{A}_{i-j}/\mathcal{B}_{i-j}/\mathcal{U}_{i-j}$ denote CONSTANT/ALL/BALANCE/UNKNOWN from bit number i to bit number j respectively counting from the most significant bit of the branch. Both of these integral distinguishers have been verified experimentally using a set of 256 randomly generated keys.

6.3.2 Integral Cryptanalysis of $3\frac{2}{7}$-Round Bel-T-256

In this section, we present our Integral attack on $3\frac{2}{7}$-round Bel-T-256 by appending one round and two S-box layers on the above derived integral distinguisher $IC1$ as illustrated in Figure 6.2.

Data Collection. We select m structures of plaintexts. In each structure, the 9 bits (bit number 18 in branch C^0 and bits 0-7 in branch D^0) vary through all 2^9 possible values and all other bits are fixed to an arbitrary constant value.

This ensures that each structure satisfies the required input division property of the integral distinguisher $IC1$. After that, we query the encryption oracle to obtain the corresponding ciphertexts. Subsequently, we apply the following key recovery procedure.

Key Recovery. For ciphertexts in each structure obtained in the data collection phase, we apply the following procedure:

1. Guess K_8 and K_4 and partially decrypt the ciphertext to obtain b_2.

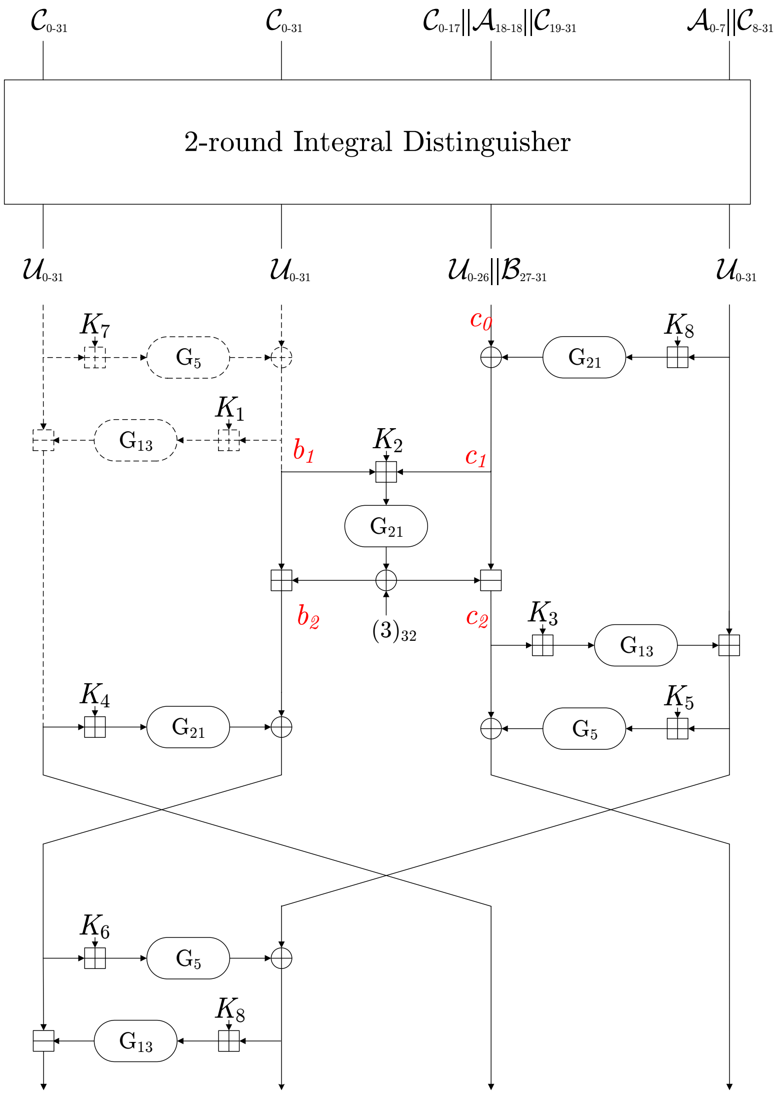
$\mathcal{C}_{0\text{-}31}$
$\mathcal{C}_{0\text{-}31}$
$\mathcal{C}_{0\text{-}17}||\mathcal{A}_{18\text{-}18}||\mathcal{C}_{19\text{-}31}$
$\mathcal{A}_{0\text{-}7}||\mathcal{C}_{8\text{-}31}$
2-round Integral Distinguisher
$\mathcal{U}_{0\text{-}31}$
$\mathcal{U}_{0\text{-}31}$
$\mathcal{U}_{0\text{-}26}||\mathcal{B}_{27\text{-}31}$
$\mathcal{U}_{0\text{-}31}$
K_7
G_5
c_0
G_{21}
K_8
K_1
G_{13}
b_1
K_2
c_1
G_{21}
b_2
$(3)_{32}$
c_2
K_3
G_{13}
K_4
G_{21}
K_5
G_5
K_6
G_5
K_8
G_{13}

Figure 6.2: $3\frac{2}{7}$-round attack on Bel-T-256

2. Guess K_6 and K_5 and partially decrypt the ciphertext to obtain c_2.

3. Recall that $b_1 = b_2 \boxminus G_{21}(b_1 \boxplus c_1 \boxplus K_2) \oplus (3)_{32}$ and $c_1 = c_2 \boxplus G_{21}(b_1 \boxplus c_1 \boxplus K_2) \oplus (3)_{32}$. Hence $b_1 \boxplus c_1 = b_2 \boxplus c_2$. Therefore, by guessing K_2, we can deduce $G_{21}(b_1 \boxplus c_1 \boxplus K_2) = G_{21}(b_2 \boxplus c_2 \boxplus K_2)$ and then compute c_1 from b_2 and c_2.

4. Guess K_3 and use the previous guessed value of K_8 to compute c_0 from c_1 and c_2.

5. For each bit in the 5 least significant bits of the 32-bit word c_0, check that its XOR sum over the structure is zero. The probability that all these 5 bits are balanced is

2^{-5}. Therefore the probability that a key is survived after this test is also 2^{-5}. This means that the number of 192-bit key candidates passed this check is $2^{192} \times 2^{-5}$.

After repeating the above procedure for m structures, the number of surviving 192-bit key candidates will be $2^{192} \times (2^{-5})^m = 2^{192-5m}$. After that, we recover the 256-bit master key by testing the 2^{192-5m} 192-bit surviving key candidates along with the remaining 2^{64} values for K_1 and K_7 using 2 plaintext/ciphertext pairs.

Attack complexity. The data complexity of the above attack is $m \times 2^9$ chosen plaintexts. The dominant part of time complexity is coming from deducing 192-bit key candidates after checking m structures. This part is equal to $\frac{7}{23} \times 2^9 \times 2^{192} \times [1 + 2^{-5} + (2^{-5})^2 + \cdots + (2^{-5})^{m-1}] = \frac{7}{23} \times 2^{201} \times \dfrac{1-(2^{-5})^m}{1-2^{-5}}$. Additionally, the part due to exhaustively searching for the master key which is equal to $2 \times 2^{64} \times 2^{192-5m} = 2^{257-5m}$. To balance the attack between data and time complexities, we take $m = 16$. This means that the data complexity will be $16 \times 2^9 = 2^{13}$ chosen plaintexts and the time complexity will be $\frac{7}{23} \times 2^{201} \times \dfrac{1-2^{-80}}{1-2^{-5}} + 2^{177} \approx 2^{199.33}$ encryption operations.

It should be noted that other choices of m can lead to possible data and time trade-off. For example, if we set $m = 1$, the data complexity will be reduced to 2^9 chosen plaintexts at the expense of increasing the time complexity to 2^{252}.

6.3.3 Integral Cryptanalysis of $3\frac{6}{7}$-Round Bel-T-256

In this section, we present our integral attack on $3\frac{6}{7}$-round Bel-T-256 by appending one round and six S-box layers on the above derived integral distinguisher $IC2$, which is the only distinguisher makes the attack feasible, as illustrated in Figure 6.3.

Data Collection. We select m structures of plaintexts. In each structure, the 32 bits (bits 11-26 in branch C^0 and bits 0-15 in branch D^0) vary through all 2^{32} possible values

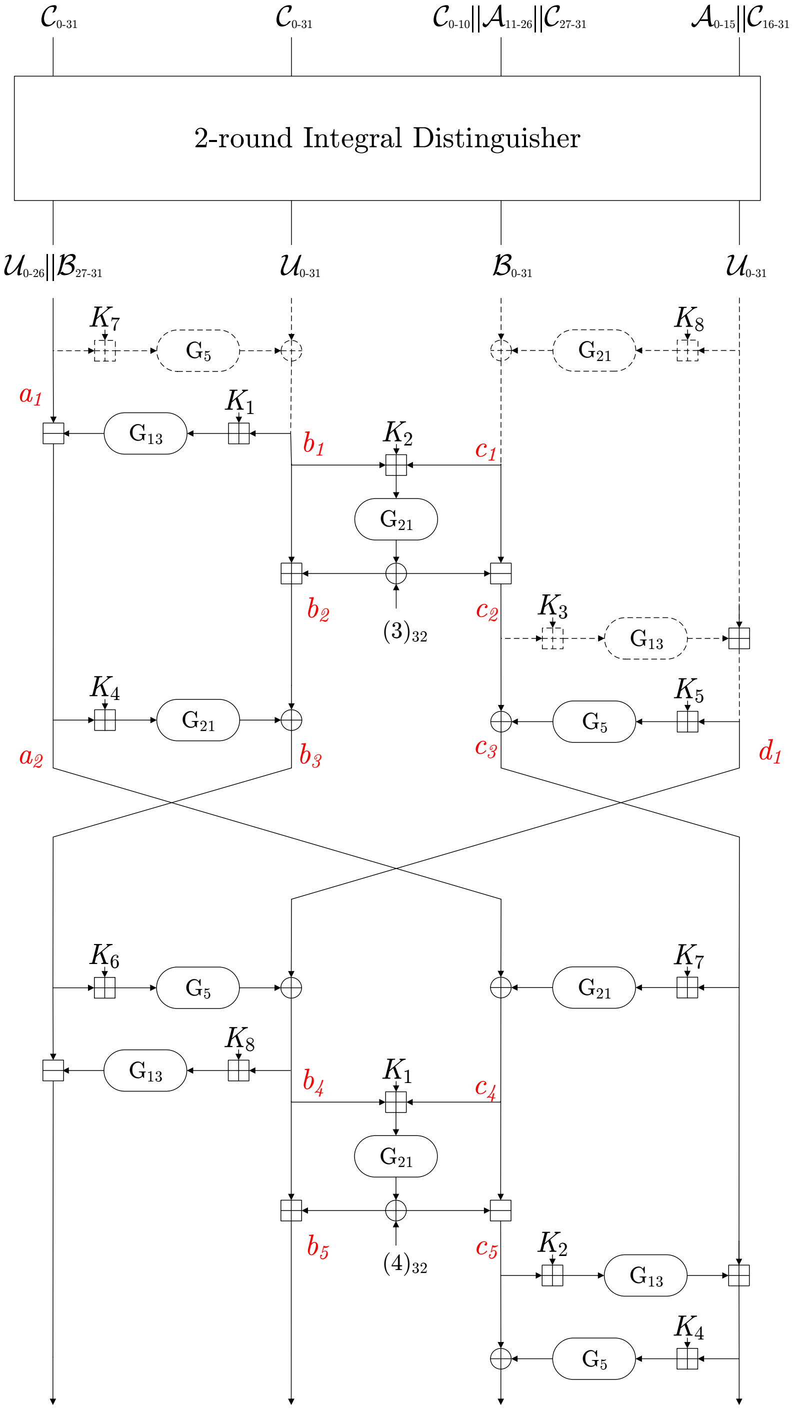
$\mathcal{C}_{0\text{-}31}$
$\mathcal{C}_{0\text{-}31}$
$\mathcal{C}_{0\text{-}10}||\mathcal{A}_{11\text{-}26}||\mathcal{C}_{27\text{-}31}$
$\mathcal{A}_{0\text{-}15}||\mathcal{C}_{16\text{-}31}$
2-round Integral Distinguisher
$\mathcal{U}_{0\text{-}26}||\mathcal{B}_{27\text{-}31}$
$\mathcal{U}_{0\text{-}31}$
$\mathcal{B}_{0\text{-}31}$
$\mathcal{U}_{0\text{-}31}$
K_7
G_5
G_{21}
K_8
a_1
G_{13}
K_1
b_1
K_2
c_1
G_{21}
b_2
$(3)_{32}$
c_2
K_3
G_{13}
K_4
G_{21}
G_5
K_5
a_2
b_3
c_3
d_1
K_6
G_5
G_{21}
K_7
G_{13}
K_8
b_4
K_1
c_4
G_{21}
b_5
$(4)_{32}$
c_5
K_2
G_{13}
K_4
G_5

Figure 6.3: $3\frac{6}{7}$-round attack on Bel-T-256

and all other bits are fixed to an arbitrary constant value. This ensures that each structure satisfies the required input division property of the integral distinguisher $IC2$. After that, we query the encryption oracle to obtain the corresponding ciphertexts. Subsequently, we apply the following key recovery procedure.

Key Recovery. For ciphertexts in each structure obtained in the data collection, we apply the following procedure:

1. Guess K_4 and partially decrypt the ciphertext to obtain c_5.
2. Recall that $b_4 = b_5 \boxminus G_{21}(b_4 \boxplus c_4 \boxplus K_1) \oplus (4)_{32}$ and $c_4 = c_5 \boxplus G_{21}(b_4 \boxplus c_4 \boxplus K_1) \oplus (4)_{32}$, hence $b_4 \boxplus c_4 = b_5 \boxplus c_5$. Therefore, by guessing K_1, we can deduce $G_{21}(b_4 \boxplus c_4 \boxplus K_1) = G_{21}(b_5 \boxplus c_5 \boxplus K_1)$ and then compute b_4 and c_4 from b_5 and c_5.
3. Guess K_2, K_6, K_7 and K_8 and deduce each 32-bit words a_2, b_3, c_3 and d_1.
4. Use the previous guessed value of K_4 to get the value of b_2 from a_2 and b_3.
5. Guess K_5 and get the value of c_2 from c_3 and d_1.
6. Recall that $b_1 = b_2 \boxminus G_{21}(b_1 \boxplus c_1 \boxplus K_2) \oplus (3)_{32}$ and $c_1 = c_2 \boxplus G_{21}(b_1 \boxplus c_1 \boxplus K_2) \oplus (3)_{32}$, hence $b_1 \boxplus c_1 = b_2 \boxplus c_2$. Therefore, by guessing K_2, we can deduce $G_{21}(b_1 \boxplus c_1 \boxplus K_2) = G_{21}(b_2 \boxplus c_2 \boxplus K_2)$ and then compute b_1 from b_2 and c_2.
7. Use the previous guessed value of K_1 to compute a_1 from a_2 and b_1.
8. For each bit in the 5 least significant bits of 32-bit word a_1, check that the XOR sum of it over the structure is zero. The probability that all these 5 bits are balanced is 2^{-5}. Therefore the probability that a key is survived after this test is also 2^{-5}. This means that the number of 224-bit key candidates passed this check is $2^{224} \times 2^{-5}$.

After repeating the above procedure for m structures, the number of surviving 224-bit key candidates will be $2^{224} \times (2^{-5})^m = 2^{224-5m}$. After that we recover the 256-bit master key by testing the 2^{224-5m} 192-bit surviving key candidates along with the remaining 2^{32} values for K_3 using 2 plaintext/ciphertext pairs.

Attack complexity. The data complexity is $m \times 2^{32}$ chosen plaintexts. The dominant part of time complexity is coming from deducing 224-bit key candidates after checking m structure. This part is equal to $\frac{10}{27} \times 2^{32} \times 2^{224} \times [1 + 2^{-5} + (2^{-5})^2 + \cdots + (2^{-5})^{m-1}] = \frac{10}{27} \times 2^{256} \times \dfrac{1-(2^{-5})^m}{1-2^{-5}}$. Additionally, the part due to exhaustively searching for the master key which is equal to $2 \times 2^{32} \times 2^{224-5m} = 2^{257-5m}$. To balance the attack between data and time complexities, we take $m = 2$. This means that the data complexity will be $2 \times 2^{32} = 2^{33}$ chosen plaintexts and the time complexity will be $\frac{10}{27} \times 2^{256} \times \dfrac{1-2^{-10}}{1-2^{-5}} + 2^{247} \approx 2^{254.61}$ encryption.

6.4 Summary

In this chapter, we investigated the security of Bel-T-256 against integral attacks based on the bit-based division property. In particular, we have built a MILP model for the Bel-T round function to automate the search for integral distinguishers based on the bit-based division property. Using two of the obtained integral distinguishers, we presented attacks on $3\frac{2}{7}$ and $3\frac{6}{7}$ rounds of Bel-T-256 with data and time complexities of 2^{13}, 2^{33} chosen plaintexts and $2^{199.33}$, $2^{254.61}$ encryption operations, respectively.

Chapter 7

Integral Cryptanalysis of Reduced-Round Tweakable TWINE

Tweakable TWINE (T-TWINE) [81] is the first lightweight dedicated tweakable block cipher family built on Generalized Feistel Structure (GFS). T-TWINE family is an extension of the conventional block cipher TWINE [94] with minimal modification by adding a simple tweak based on the SKINNY's tweakey schedule [6]. Similar to TWINE, T-TWINE has two variants, namely T-TWINE-80 and T-TWINE-128. The two variants have the same block size of 64 bits and a variable key length of 80 and 128 bits. In this chapter, we study the implications for adding the tweak on the security of T-TWINE against integral cryptanalysis. In particular, we first utilize the bit-based division property to search for the longest integral distinguisher. As a result, we are able to perform a distinguishing attack against 19 rounds using $2^6 \times 2^{63} = 2^{69}$ chosen tweak-plaintext combinations. We then convert this attack to key recovery attacks against 26 and 27 rounds (out of 36) of T-TWINE-80 and T-TWINE-128, respectively. By prepending one round before the distinguisher and using dynamically chosen plaintexts, we manage to extend the attack one more round without using the full codebook of the plaintext. Therefore, we are able to attack 27 and 28 rounds of T-TWINE-80 and T-TWINE-128, respectively.

7.1 Introduction

A Tweakable block cipher (TBC) is a symmetric-key cryptographic primitive that takes an auxiliary input called *tweak* in addition to the inputs of traditional block ciphers, plaintext message and cryptographic key [66]. Ideally, a different tweak value gives randomly chosen and different instant of the permutation over the message space without needing to change the key which may be costly in traditional block ciphers. A Tweakable block cipher is a powerful primitive that can be used in several applications such as disk encryption in which the repeated same plaintext should be encrypted to different ciphertexts under the same key. The concept of tweakable block ciphers also allows interesting modes for authenticated encryption such as OCB3 [60] and Counter-in-Tweak [75].

There are two general approaches to build TBCs: (i) using ordinary block ciphers through modes of operation, and (ii) dedicated constructions. Both the LRW and XEX modes of operations [80] are examples of the first approach. For a block cipher with n-bit block, the security of these modes is guaranteed up to around $2^{n/2}$ queries. For a higher level of security, we can use a dedicated TBC that is built with the tweak concept from the beginning such as Deoxys-BC [52], SKINNY [6], and CRAFT [7].

Tweakable TWINE (T-TWINE) [81] is the first lightweight dedicated TBC that is built on Generalized Feistel Structure (GFS). It was built with the goal of reducing the cost of design, security evaluation, and implementation. Therefore, the designers decided to reuse a well-designed GFS block cipher, TWINE [94], and attached an extremely simple tweak scheduling to it. Similar to TWINE, T-TWINE has two variants namely, T-TWINE-80 and T-TWINE-128. These variants have the same block size of 64 bits, a tweak of 64 bits, and a variable key length of 80 and 128 bits.

The security of T-TWINE is evaluated by its designers against distinguishing attacks including differential, linear, impossible differential, and integral cryptanalysis. Regarding integral cryptanalysis, they only reported an 11-round integral distinguisher. Key recovery attacks based on impossible differential against reduced-round of T-TWINE are presented in [100].

In this chapter, we study the security of T-TWINE against integral attacks. More precisely,

1. We utilize the MILP models of the bit-based division property to search for the longest integral distinguisher in the chosen tweak, chosen tweak-plaintext, and chosen tweak-ciphertext attack settings. As a result, we found two 11-round integral distinguishers using a tweak with only one active nibble in the chosen tweak setting. We also checked the 11-round distinguisher reported in the design paper and we show that it is not correct. All the found 11-round distinguishers are verified experimentally. Furthermore, we found several 19-round integral distinguishers in both chosen tweak-plaintext and chosen tweak-ciphertext settings. This allows us to attack an extra three rounds more than TWINE which has 16-round integral distinguisher [105]. The best distinguishing attack can be performed using $2^6 \times 2^{63} = 2^{69}$ chosen tweak-plaintext combinations.

2. We employ meet-in-the-middle [83] and partial-sum [40] techniques to convert the best distinguishing attack to key recovery attacks against 26 (27) out of 36 rounds of T-TWINE-80 (T-TWINE-128) by appending 7 (8) rounds after the disntinguisher.

3. By prepending one round before the distinguisher and using dynamically chosen plaintexts [21], we managed to extend the attack one more round without using the full codebook of the plaintext. Therefore, we are able to attack 27 and 28 rounds of T-TWINE-80 and T-TWINE-128, respectively.

Table 7.1 summarizes the complexities of our attacks and contrast them with the complexities of the impossible differential attacks presented in [100].

The rest of this chapter is organized as follows. In Section 7.2, we briefly revisit the specifications of T-TWINE. The detailed integral distinguishing attacks against T-TWINE

Table 7.1: Attack results on T-TWINE where CTP denotes chosen tweak-plaintext.

	Attack	#Rounds	Data	Time	Memory	Reference
T-TWINE-80	Imp. diff.	25	$2^{65.5}$ CTP	$2^{70.86}$	2^{66}	[100]
	Integral	26	$2^{70.58}$ CTP	$2^{72.62}$	$2^{67.62}$	Sec. 7.4.1
		27	$2^{70.95}$ CTP	$2^{75.79}$	$2^{71.08}$	Sec. 7.5.1
T-TWINE-128	Imp. diff.	27	2^{64} CTP	$2^{120.83}$	2^{118}	[100]
	Integral	27	$2^{71.58}$ CTP	$2^{109.54}$	$2^{90.58}$	Sec. 7.4.2
		28	$2^{72.27}$ CTP	$2^{113.38}$	$2^{94.32}$	Sec. 7.5.1

is explained in Section 7.3. In Section 7.4, we describe the key recovery attacks against 26 and 27 rounds of T-TWINE-80 and T-TWINE-128, respectively. Then, the details of our attacks against 27 and 28 rounds of T-TWINE-80 and T-TWINE-128 using dynamically chosen plaintexts are presented in Section 7.5. Finally, the chapter is concluded in Section 7.6.

7.2 T-TWINE Specifications

The following notation is used throughout the rest of the chapter:

- K: The 80 or 128 bits master key.
- K_j: The j^{th} nibble of K. The indices of the nibbles begin from 0.
- RK^i: The 32-bit round key used in round i.
- RK^i_j: The j^{th} nibble of RK^i. The indices of the nibbles begin from 0.
- T: The 64-bit tweak.
- T_j: The j^{th} nibble of the tweak T.
- RT^i: The 24-bit round tweak used in round i, where $RT^i \leftarrow t^i_0||t^i_1||$ $t^i_2||t^i_3||t^i_4||t^i_5$, and t^i_j is the j^{th} nibble of RT^i.

Table 7.2: Nibble shuffle π

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

- X^i: The 16 nibbles input to round i. The indices of the round begin from 1.
- X^i_j: j^{th} nibble of X^i.
- $x[m]$: m^{th} bit of the nibble x where $x[0]$ is the least significant bit.
- $\oplus$: The XOR operation.
- $||$: The concatenation operation.
- $Rotz(x)$: The z-bit left cyclic shift of x.

As we mentioned above, T-TWINE is an extension of the conventional block cipher TWINE. It takes a tweak of 64 bits as an extra input in addition to a block of plaintext with 64 bits in order to produce a block of ciphertext using 80 or 128 bits of a secret key. T-TWINE structure consists of three parts: data processing which is a slightly modified version of the equivalent part in TWINE to deal with the extra input, key scheduling function of TWINE, and tweak scheduling function. The two variants of T-TWINE are the same except in the key scheduling function.

Data Processing. The round function is based on a variant of Type-2 GFS [93] with 16 4-bit nibbles as depicted in Figure 7.1. It consists of a nonlinear layer (F-function operations), round tweak XOR, and a diffusion layer which is a 16-nibble shuffle operation (π, see Table 7.2). The F-function operation is a round-key XOR followed by 4-bit S-box (S, see Table 7.3). This round function is iterated 36 times in both variants where the diffusion layer is omitted from the last round.

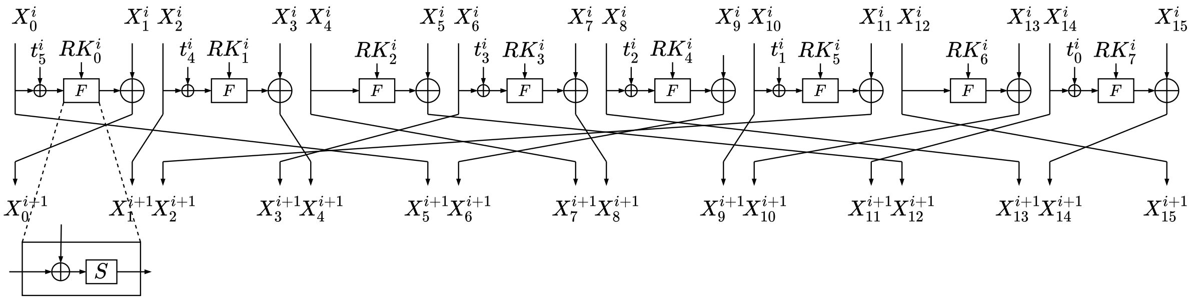

Figure 7.1: T-TWINE round function

Table 7.3: 4-bit S-box (S) of T-TWINE in hexadecimal form

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

Key Scheduling Function. Each variant of T-TWINE has its own key schedule. The key scheduling function is used to stretch 80/128 bits of the master key K to 36 32-bit round keys RK^i where $1 \leq i \leq 36$. Algorithms 2 and 3 show the details of these key schedules. For more details, see [81, 94].

Tweak Scheduling Function. A 64-bit tweak T is used to generate 36 24-bit round tweaks RT^i where $1 \leq i \leq 36$ using a permutation-based function. Firstly, the 64-bit tweak T is loaded to 16 4-bit nibbles t_j^1 where $0 \leq j \leq 15$. In i-th round, the first 6 nibbles $(t_0^i, \ldots, t_5^i)$ are used as the round tweak RT^i, then these nibbles are shuffled using a 6-nibble permutation π^t, s.t. $(0, 1, 2, 3, 4, 5) \rightarrow (1, 0, 4, 2, 3, 5)$. After that, all nibbles are shifted by 6 nibbles to construct t_j^{i+1} where $0 \leq j \leq 15$ as depicted in Figure 7.2.

7.3 Integral Distinguishing Attacks

Since T-TWINE is an extension of TWINE which has 16-round integral distinguisher using 2^{63} chosen plaintexts [105], in this section we study the effect of the freedom gained by adding a tweak to the structure. Thereby, we report the result regarding the integral distinguishers in the three attack settings: chosen tweak, chosen tweak-plaintext, and

Algorithm 2: Key Schedule of T-TWINE-80

```
Data: The 80-bit master key $K$
Result: The round keys $RK = RK^1||RK^2||\cdots||RK^{36}$
$k_0||k_1||\cdots||k_{19} \leftarrow K$;
for $i \leftarrow 1$ to 35 do
    $RK^i \leftarrow k_1||k_3||k_4||k_6||k_{13}||k_{14}||k_{15}||k_{16}$;
    $k_1 \leftarrow k_1 \oplus S(k_0)$;
    $k_4 \leftarrow k_4 \oplus S(k_{16})$;
    $k_7 \leftarrow k_7 \oplus (0||CON_H^i)$;
    $k_{19} \leftarrow k_{19} \oplus (0||CON_L^i)$;
    $k_0||\cdots||k_3 \leftarrow Rot4(k_0||\cdots||k_3)$;
    $k_0||\cdots||k_{19} \leftarrow Rot16(k_0||\cdots||k_{19})$;
end
$RK^{36} \leftarrow k_1||k_3||k_4||k_6||k_{13}||k_{14}||k_{15}||k_{16}$;
$RK \leftarrow RK^1||RK^2||\cdots||RK^{36}$;
```

Algorithm 3: Key Schedule of T-TWINE-128

```
Data: The 128-bit master key $K$
Result: The round keys $RK = RK^1||RK^2||\cdots||RK^{36}$
$k_0||k_1||\cdots||k_{31} \leftarrow K$;
for $i \leftarrow 1$ to 35 do
    $RK^i \leftarrow k_2||k_3||k_{12}||k_{15}||k_{17}||k_{18}||k_{28}||k_{31}$;
    $k_1 \leftarrow k_1 \oplus S(k_0)$;
    $k_4 \leftarrow k_4 \oplus S(k_{16})$;
    $k_{23} \leftarrow k_{23} \oplus S(k_{30})$;
    $k_7 \leftarrow k_7 \oplus (0||CON_H^i)$;
    $k_{19} \leftarrow k_{19} \oplus (0||CON_L^i)$;
    $k_0||\cdots||k_3 \leftarrow Rot4(k_0||\cdots||k_3)$;
    $k_0||\cdots||k_{31} \leftarrow Rot16(k_0||\cdots||k_{31})$;
end
$RK^{36} \leftarrow k_2||k_3||k_{12}||k_{15}||k_{17}||k_{18}||k_{28}||k_{31}$;
$RK \leftarrow RK^1||RK^2||\cdots||RK^{36}$;
```

chosen tweak-ciphertext. To this end, we utilize the MILP models of the propagation rules of the bit-based division property described in the previous chapter (Section 6.2.1) to automate the search process using Gurobi optimizer [48]. We obtain the best distinguisher in two steps. In the first step, we look for a distinguisher that covers the maximum number of rounds irrespective of the data complexity. Then, we try to reduce the data complexity of the longest one in the second step. We use the following notation to present the status

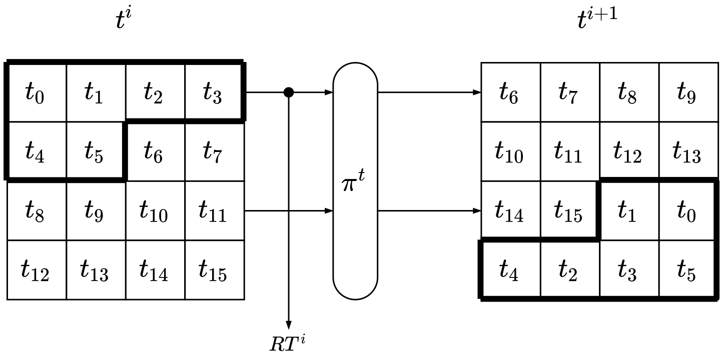
t^i
t^{i+1}
t_0 t_1 t_2 t_3
t_4 t_5 t_6 t_7
t_8 t_9 t_{10} t_{11}
t_{12} t_{13} t_{14} t_{15}
π^t
RT^i
t_6 t_7 t_8 t_9
t_{10} t_{11} t_{12} t_{13}
t_{14} t_{15} t_1 t_0
t_4 t_2 t_3 t_5

Figure 7.2: Tweak schedule of T-TWINE

of each nibble of the tweak, plaintext, and ciphertext:

- $\boxed{\mathcal{C}}$ each bit of the nibble is fixed to constant.
- $\boxed{\mathcal{A}}$ all bits of the nibble are active.
- $\boxed{\tilde{\mathcal{A}}}$ all bits of the nibble are active except one arbitrary bit is constant.
- $\boxed{\mathcal{B}}$ each bit of the nibble is balanced (the XOR sum is zero).
- $\boxed{\mathcal{U}}$ a nibble with unknown status.

Chosen tweak setting. In this setting, all the plaintext bits are fixed to constant values and some or all the bits of the tweak are active while the remaining bits are constant.

In the first step, we set all bits of the tweak to active. We then target r rounds and use our MILP model to search for some balanced bits. If there is at least one balanced bit, we increase the target rounds to $r+1$ and repeat the search process in the same way. Otherwise, we conclude that the disnguisher with the maximum number of rounds based on our model covers r rounds. Based on our evaluation, there is no distinguisher for 12 or more rounds and the longest distinguisher is an 11-round one. In the second step, we try to reduce the data complexity of that 11-round distinguisher by minimizing the number of active nibbles in the tweak. To this end, we start with only one active nibble and if there is no balanced bits, we progressively increase the number of active nibbles. Fortunately, we find two distinguishers with only one active nibble as shown bellow:

Plaintext	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$																	
Tweak	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{A}$	$\mathcal{C}$	$\xrightarrow{11R}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{B}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$
Tweak	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{A}$	$\xrightarrow{11R}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{B}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{B}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$

It should be mentioned that the designers have reported in [81] a different 11-round integral distinguisher in which the plaintext nibbles are fixed to constant, the three nibbles (5, 10, 11) in the tweak are actives, and the remaining nibbles in the tweak are fixed to constant. This distinguisher has two balanced nibbles (0, 11) in the ciphertext side as shown below. However, when we test this distinguisher using our MILP model with the same input settings, we confirmed that there is only one balanced nibble (11) in the ciphertext side.

Plaintext	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$																		
Tweak	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{A}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\xrightarrow{11R}$	$\mathcal{B}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{B}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	✗ ([81])
Tweak	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{A}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{A}$	$\mathcal{A}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\mathcal{C}$	$\xrightarrow{11R}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{B}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	$\mathcal{U}$	✓(Ours)

Since the data complexity for each one of the two 11-round integral distinguishers we have proposed is 2^4, we have verified the correctness of them experimentally to validate our results. Additionally, the data complexity of the 11-round distinguisher with the same input settings as the distinguisher reported in [81] is 2^{12}, we also have verified experimentally that it has only one balanced nibble (11) in the ciphertext side which is consistent with the result using our MILP model[*].

Chosen tweak-plaintext setting. In this setting, some of plaintext bits are active and the remaining bits are constant. For the tweak, some or all bits are active and the remaining bits are constant.

Since the goal of the first step is to obtain the longest distinguisher, we set the

[*] The code can be found at:
https://github.com/mhgharieb/Integral-Attack-T-TWINE

64 bits of the tweak and 63 bits of the plaintext to active and the remaining bit of the plaintext to constant[†]. We then target r rounds and iterate over the 64 positions of the constant bit until we find some balanced bits or terminate without finding any. In the first case, we increase the target rounds to $r+1$ and repeat the search process in the same way. Otherwise, we conclude that the disnguisher with the maximum number of rounds based on our model covers r rounds. In our evaluation, we found that the 19-round distinguisher is the longest one.

In order to convert the distinguishing attack to a key recovery attack applicable for both variants T-TWINE-80 and T-TWINE-128, the data complexity of the distinguisher must be less than 2^{80}. Therefore, we limit the search process to find a distinguisher that needs up to 80 active bits.

During the second step, we try to reduce the data complexity by minimizing the number of active bits in both plaintext and tweak. We follow the technique described in [89] to reduce the active bits of the plaintext. In particular, we repeat the previous step for 19 rounds and instead of stopping the search process if there are some balanced bits, we keep a record of the position of the constant bit in case of no balanced bits. In our evaluation, there are 32 bits corresponding to the nibbles $(1, 3, 5, 7, 9, 11, 13, 15)$ that must be active to obtain 19-round distinguisher and the remaining bits may be active or constant. After that, we try all the combinations of 2 out of 32 bits that might be constant and check if the 19-round distinguisher exists. Unfortunately, such distinguisher does not exist if we set any two bits in the plaintext to constant. Regarding the active bits reduction in the tweak, we start with only one active nibble and if there is no distinguisher, we progressively increase the number of active nibbles.

In our evaluation, there are several 19-round integral distinguishers using tweak with two active nibbles. Moreover, we are able to reduce the active bits to 7 bits for some of them and 6 bits for the distingiusher that we will use during the key recovery attacks.

[†] The data complexity of plaintext must be less than the full codebook because using the full codebook of any permutation (a random permutation or a block cipher) always gives a balanced output.

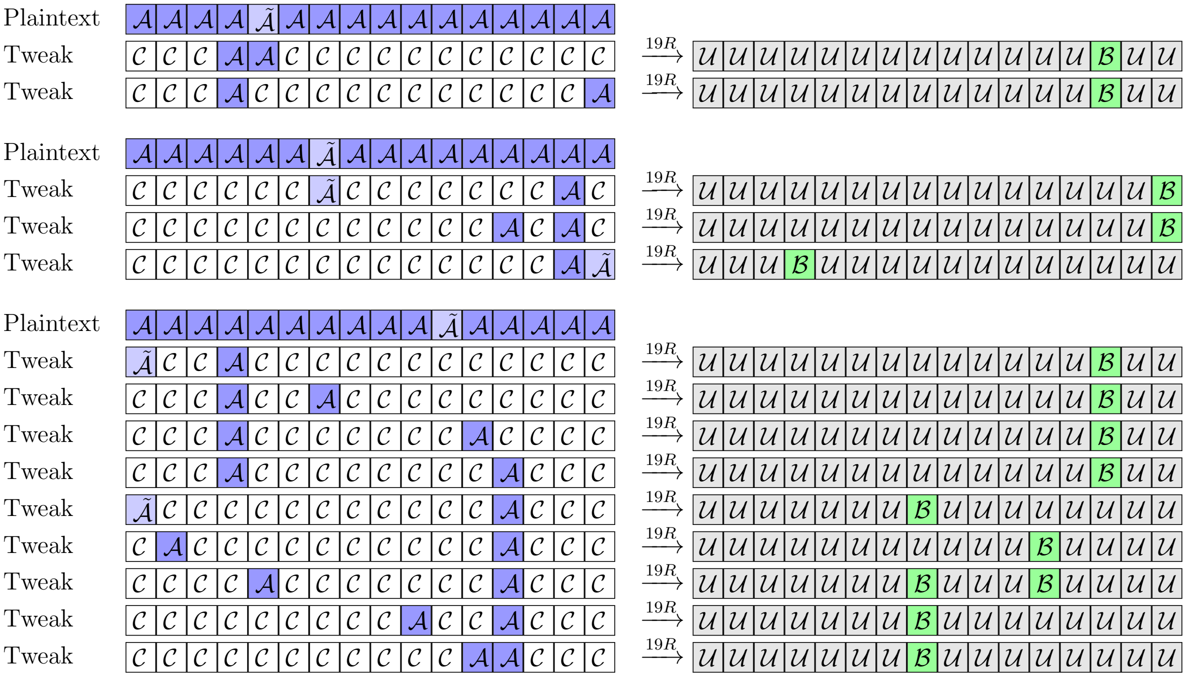

Figure 7.3: 104 19-round integral distinguishers in chosen tweak-plaintext setting, where the three groups consist of $4 \times (1+1) = 8$, $4 \times (4+1+4) = 36$, and $4 \times (4+1+1+1+4+1+1+1+1) = 60$ distinguishers.

Figure 7.3 summarizes 40 distinguishers with $2^8 \times 2^{63} = 2^{71}$, and 64 distinguishers with $2^7 \times 2^{63} = 2^{70}$ chosen tweak-plaintext combinations.

Chosen tweak-ciphertext setting. In this setting, some of ciphertext bits are active and the remaining bits are constant. For the tweak, some or all bits are active and the remaining bits are constant.

We followed the same technique we have used in chosen tweak-plaintext setting and we found that the 19-round integral distinguisher is the longest one. Like chosen tweak-plaintext setting, the distinguisher does not exist if there are two constant bits in the ciphertext. Also, there are several two active nibbles combinations of the tweak that lead to 19-round distinguisher. Moreover, we are able to reduce, for some of them, the active bits to only 7. Figure 7.4 summarizes 104 19-round integral distinguishers, 64 of them need $2^7 \times 2^{63} = 2^{70}$ chosen tweak-ciphertext combinations and the remaining need $2^8 \times 2^{63} = 2^{71}$ chosen tweak-ciphertext combinations.

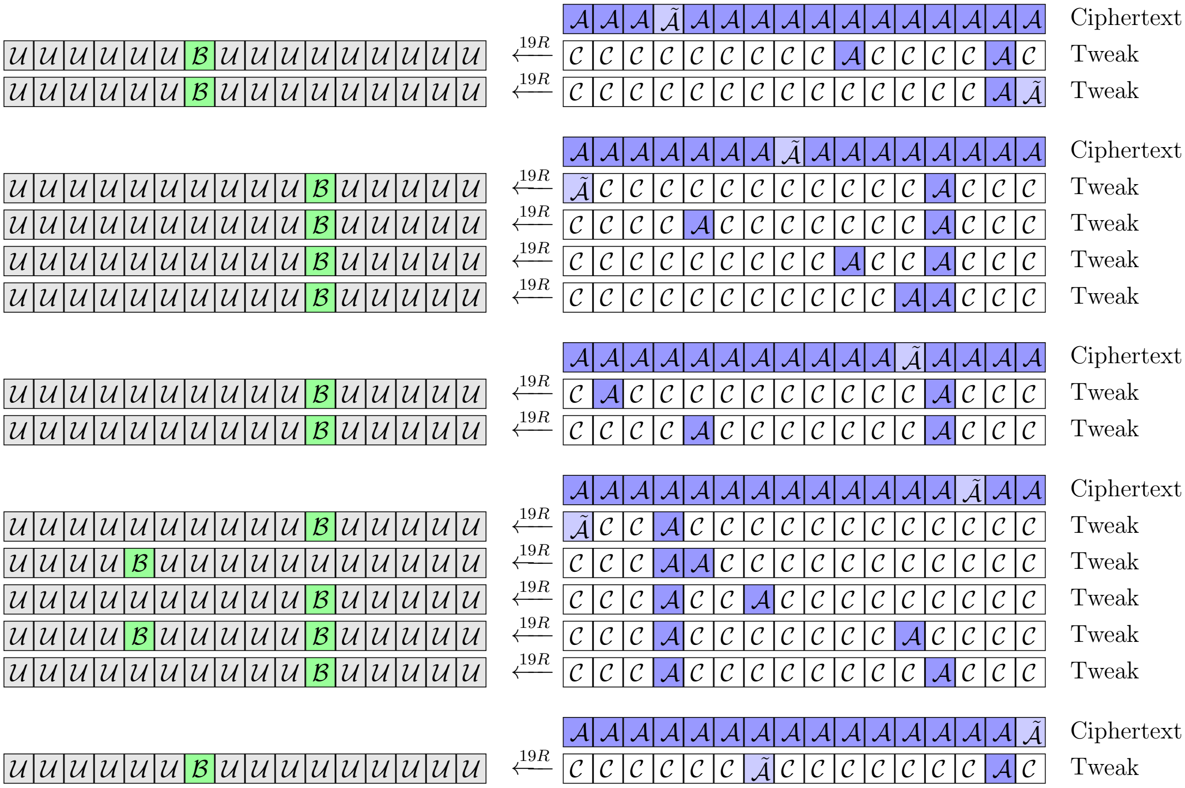

Figure 7.4: 104 19-round Integral distinguishers in chosen tweak-ciphertext setting, where the five groups consist of 20, 28, 8, 32, and 16 distinguishers.

7.4 Integral Attacks on T-TWINE

We convert the distinguishing attacks described in the previous section to key recovery attacks against reduced-round versions of T-TWINE. In particular, we target 26 and 27 rounds of T-TWINE-80 and T-TWINE-128, respectively, using the following 19-round distinguisher that needs 6 and 63 active bits of the tweak and the plaintext, respectively:

$$\text{Plaintext}: (\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A}_3,\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A},\mathcal{A})$$

$$\text{Tweak}: (\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{A}_{1,3},\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{C},\mathcal{A},\mathcal{C})$$

$$\downarrow 19R$$

$$(\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{U},\mathcal{B})$$

where $\mathcal{A}_3$ means all bits of the nibble are active except bit 3, counted from the least significant bit, is constant and $\mathcal{A}_{1,3}$ means bits (0 and 2) are active and bits (1 and 3) are constant.

In the following, we revisit the Meet-in-the-Middle technique [83] and Partial-Sum technique [40] that we use to enhance the time complexities of our proposed attacks.

Meet-in-the-Middle Technique. Let $Z_j^i, (0 \leq j \leq 7)$ denote the output of the F functions in i-th round of T-TWINE. Consider the 19-round distinguisher mentioned above, then the nibble X_{15}^{20} is balanced ($\bigoplus X_{15}^{20} = 0$). Since this nibble can be expressed as a linear combination of Z_7^{20} and X_{14}^{21}, we can obtain the following relation

$$\bigoplus Z_7^{20} = \bigoplus X_{14}^{21}$$

In meet-in-the-middle technique [83], each sum is independently computed from ciphertexts (*e.g.*, see Figure 7.5) and the subkeys used during the computation are stored in two different tables indexed by the value of the sum. After that, we consider the matches between the two tables, in the same manner of the meet-in-the-middle attack, as candidate subkeys because they satisfy the previous relation. Since the procedure to obtain both $\bigoplus Z_7^{20}$ and $\bigoplus X_{14}^{21}$ independently involves less number of subkeys than the one to obtain $\bigoplus X_{15}^{20}$ directly, the time complexity will be improved.

Partial-Sum Technique. Ferguson *et al.* introduced the partial-sum technique to improve the time complexity of integral attacks [40]. Suppose the key recovery procedure during the integral cryptanalysis involves N operations, κ-bit subkey and $2^{|I|}$ ciphertexts, then the time complexity of the direct computation will be $N \times 2^{|I|+\kappa}$ operations. Using the partial-sum technique, this time complexity can be improved as follows. We firstly store the ciphertexts that appear odd times in the memory whereas the ciphertexts that appear even times are discarded since they have no effect on the balanced property. Then, we guess a part of the subkey (κ_1-bit) and partially decrypt the ciphertexts through a single operation to an intermediate state with $|I_1|$-bit size (that can have up to $2^{|I_1|}$ values) such that $|I_1| \leq |I|$. The time complexity of this step is $2^{|I|+\kappa_1}$ operations. After that, we repeat the step of storing the values that appear odd times and partially decrypting the intermediate state using κ_i-bit to get another intermediate state with $|I_i|$-bit size such that $|I_i| \leq |I_{i-1}|$. The time complexity of the i-th step will be $2^{|I_{i-1}|+\kappa_1+\cdots+\kappa_i}$ where I_0 is I, and the whole time complexity will be

$$\sum_{i=1}^{N} 2^{|I_{i-1}|+\kappa_1+\cdots+\kappa_i} < \sum_{i=1}^{N} 2^{|I|+\kappa} = N \times 2^{|I|+\kappa}$$

In the following, we give the details of the key recovery attack against T-TWINE-80.

7.4.1 Attack on 26-Round T-TWINE-80

The ciphertexts of 26-round of T-TWINE-80 can be written as X^{27}. The process of obtaining $\bigoplus X^{20}_{15}$ involves the following 27 round keys (see Figure 7.5):

$$RK^{26}, RK^{25}_{[0,1,2,3,4,5,7]}, RK^{24}_{[0,1,2,6,7]}, RK^{23}_{[0,4,6]}, RK^{22}_{[4,5]}, RK^{21}_{5}, RK^{20}_{7}$$

However, we only need to guess 76 bits in 19 round keys and the other 8 round keys can be computed based on the key schedule as follows:

$$\begin{aligned}
RK^{24}_0 &= RK^{25}_7 \oplus S(RK^{26}_6 \oplus (0||CON^{25}_L)), & RK^{24}_1 &= RK^{26}_5,\\
RK^{24}_2 &= S^{-1}(RK^{26}_7 \oplus RK^{25}_0) \oplus S(RK^{24}_7), & RK^{23}_4 &= RK^{26}_0,\\
RK^{23}_6 &= RK^{26}_1 \oplus (0||CON^{25}_H), & RK^{22}_4 &= RK^{25}_0,\\
RK^{21}_5 &= RK^{26}_4, & RK^{20}_7 &= RK^{26}_6 \oplus S(RK^{26}_2) \oplus (0||CON^{25}_L).
\end{aligned}$$

where CON^{25}_L and CON^{25}_H are predefined constants.

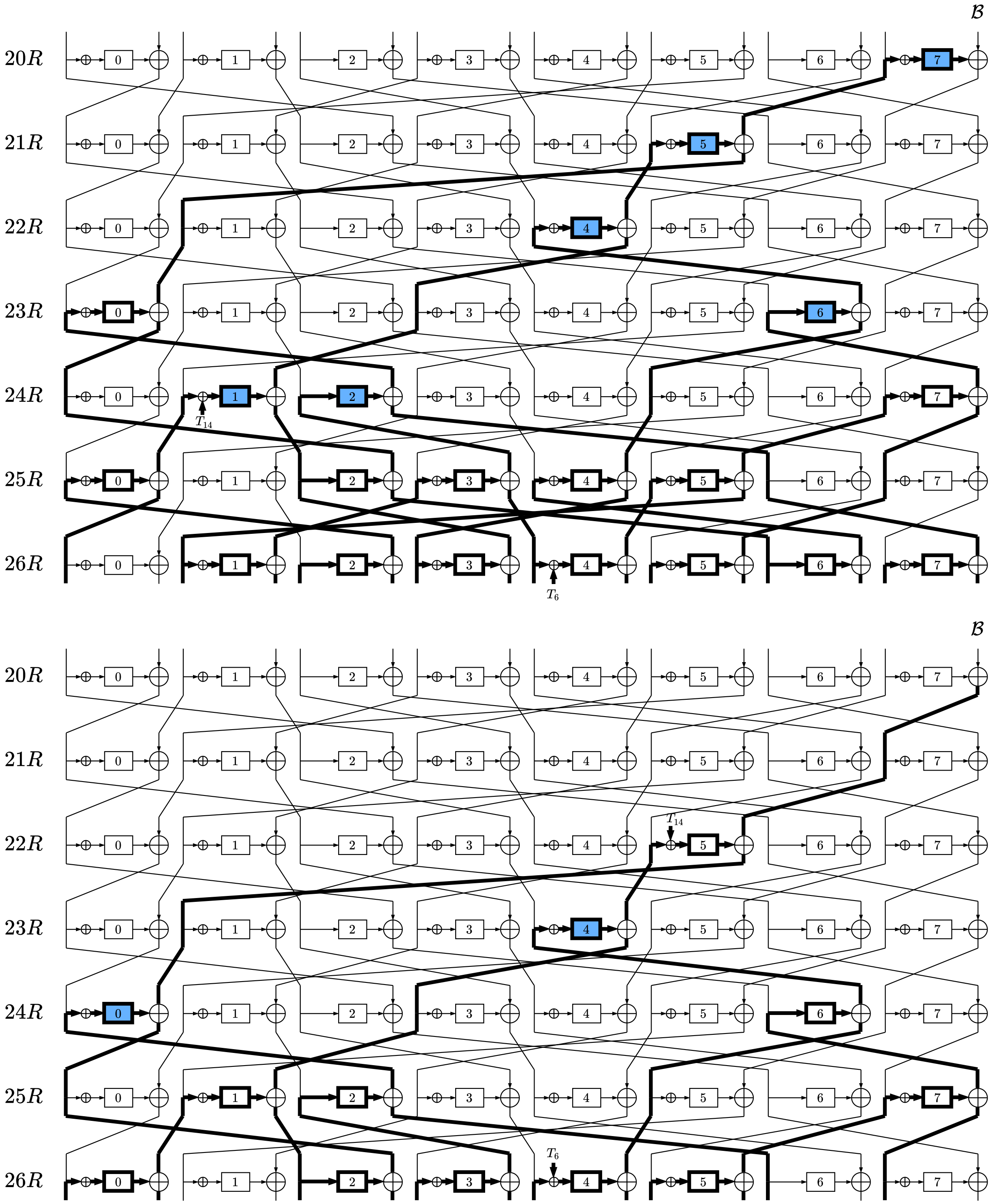

Figure 7.5: Analysis rounds of T-TWINE-80 where the upper part is used during computing $\bigoplus Z_7^{20}$ and the lower part is used during computing $\bigoplus X_{14}^{21}$.

Key Recovery Procedure. We firstly construct a data structure where all the bits of the plaintext X^1 are active except the bit $X_6^1[3]$ which is fixed to constant. For the tweak, the 6 bits $T_6[0,2]||T_{14}$ are active whereas the other bits are fixed to constant. We then ask the encryption oracle to obtain the corresponding ciphertext (X^{27}). After that, we initialize two empty hash tables H_Z and H_X with 2^{56} and 2^{40} entries to store the values of $\bigoplus Z_7^{20}$ and $\bigoplus X_{14}^{21}$, respectively, indexed by the round keys used during the computations.

Since obtaining $\bigoplus Z_7^{20}$ (the upper part of Figure 7.5) requires much more computation than obtaining $\bigoplus X_{14}^{21}$ (the lower part of Figure 7.5), we only explain the procedure to obtain $\bigoplus Z_7^{20}$. The attack starts by storing the values of $X^{27}_{[0,2,3,4,5,6,7,8,9,10,11,12,13,14,15]}||T_6[0,2]$ $||T_{14}$ that appear odd times in a list called the state $\mathcal{S}_0$ which has a size of up to 2^{66} 66-bit values. Then, we guess at the i-th step a round key (or deduce it based on the key schedule as shown above) and partially decrypt the values in the state $\mathcal{S}_{i-1}$, then store the values of the output that appear odd times in a new state $\mathcal{S}_i$. For example, we guess at step 1 RK_2^{26} and partially decrypt X_4^{27} and X_5^{27} to obtain $X_5^{26} = X_5^{27} \oplus S(X_4^{27} \oplus K_2^{26})$. The state size after compression is up to 2^{62} 62-bit values. The time complexity of this step is $2^4 \times 2^{66} = 2^{70}$ F-function operations. Table 7.4 summarizes the steps of the attack procedure.

Finally, we access the hash tables (H_Z, H_X) for each 76-bit key, and we consider a 76-bit key as a candidate if the two entries are equal. The 4 balanced bits lead to 4 bits filtration, therefore we get 2^{72} 76-bit candidates for the round keys when we use a single data structure. We can reduce the number of the candidates by repeating the attack using another data structure. Thanks to the key schedule, we can obtain 2^{76} 80-bit candidates for the master key corresponding to these 2^{72} 76-bit round keys by guessing 4-bit round key. The details of this step can be found in Appendix A. We then get the right master key by exhaustively searching over these candidates using 2 plaintext/ciphertext pairs.

Attack Complexity. When we use a single data structure, we need $2^6 \times 2^{63} = 2^{69}$ queries to the encryption oracle. From Table 7.4, we need $2^{78.13}$ F-function operations to compute $\bigoplus Z_7^{20}$. Using the same method, we need $2^{59.91}$ operations to compute $\bigoplus X_{14}^{21}$.

Table 7.4: Summary of the procedure to obtain $\bigoplus Z_7^{20}$ where 'Size' refers to the size of the intermediate state $\mathcal{S}_i$ after the partial decryption at each step, the nibbles $\boxed{X_j^r}$ in the state $\mathcal{S}_{i-1}$ are replaced by the nibbles X_j^rs in the state $\mathcal{S}_i$ during the i-th step, and 'Complexity' is measured in term of F-function operations except step 0 is measured in number of memory accesses (MA).

Step	Key	Size	The State $(\mathcal{S}_i)$	Complexity
0	-	2^{66}	$X_0^{27}, X_2^{27}, X_3^{27}, \boxed{X_4^{27}}, \boxed{X_5^{27}}, X_6^{27}, X_7^{27}, X_8^{27}, X_9^{27}, X_{10}^{27}, X_{11}^{27}, X_{12}^{27}, X_{13}^{27}, X_{14}^{27}, X_{15}^{27}, T_6, T_{14}$	$2^{66}\,MA$
1	RK_2^{26}	2^{62}	$X_0^{27}, X_2^{27}, X_3^{27}, X_5^{26}, X_6^{27}, X_7^{27}, X_8^{27}, X_9^{27}, \boxed{X_{10}^{27}}, \boxed{X_{11}^{27}}, X_{12}^{27}, X_{13}^{27}, X_{14}^{27}, X_{15}^{27}, T_6, T_{14}$	$2^4 \times 2^{66} = 2^{70}$
2	RK_5^{26}	2^{58}	$X_0^{27}, X_2^{27}, X_3^{27}, X_5^{26}, X_6^{27}, X_7^{27}, X_8^{27}, X_9^{27}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, \boxed{X_{14}^{27}}, \boxed{X_{15}^{27}}, T_6, T_{14}$	$2^8 \times 2^{62} = 2^{70}$
3	RK_7^{26}	2^{54}	$\boxed{X_0^{27}}, X_2^{27}, X_3^{27}, \boxed{X_5^{26}}, X_6^{27}, X_7^{27}, X_8^{27}, X_9^{27}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}, T_6, T_{14}$	$2^{12} \times 2^{58} = 2^{70}$
4	RK_0^{25}	2^{50}	$X_1^{25}, X_2^{27}, X_3^{27}, \boxed{X_6^{27}}, \boxed{X_7^{27}}, X_8^{27}, X_9^{27}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}, T_6, T_{14}$	$2^{16} \times 2^{54} = 2^{70}$
5	RK_3^{26}	2^{50}	$\boxed{X_1^{25}}, X_2^{27}, X_3^{27}, X_6^{26}, \boxed{X_7^{26}}, X_8^{27}, X_9^{27}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}, T_6, \boxed{T_{14}}$	$2^{20} \times 2^{50} = 2^{70}$
6	RK_1^{24}	2^{46}	$X_3^{24}, X_2^{27}, X_3^{27}, X_6^{26}, X_7^{26}, \boxed{X_8^{27}}, \boxed{X_9^{27}}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}, \boxed{T_6}$	$2^{20} \times 2^{50} = 2^{70}$
7	RK_4^{26}	2^{44}	$X_3^{24}, \boxed{X_2^{27}}, \boxed{X_3^{27}}, X_6^{26}, X_7^{26}, X_8^{26}, X_9^{26}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}$	$2^{24} \times 2^{46} = 2^{70}$
8	RK_1^{26}	2^{44}	$X_3^{24}, X_2^{26}, \boxed{X_3^{26}}, X_6^{26}, X_7^{26}, \boxed{X_8^{26}}, X_9^{26}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}$	$2^{28} \times 2^{44} = 2^{72}$
9	RK_3^{25}	2^{40}	$X_3^{24}, \boxed{X_2^{26}}, X_6^{26}, X_7^{26}, X_7^{25}, \boxed{X_9^{26}}, X_{11}^{26}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}$	$2^{32} \times 2^{44} = 2^{76}$
10	RK_5^{25}	2^{36}	$X_3^{24}, X_6^{26}, X_7^{26}, X_7^{25}, \boxed{X_{11}^{25}}, \boxed{X_{11}^{26}}, X_{12}^{27}, X_{13}^{27}, X_{15}^{26}$	$2^{36} \times 2^{40} = 2^{76}$
11	RK_7^{24}	2^{32}	$X_3^{24}, X_6^{26}, X_7^{26}, \boxed{X_7^{25}}, X_{12}^{27}, X_{13}^{27}, X_{15}^{24}, \boxed{X_{15}^{26}}$	$2^{40} \times 2^{36} = 2^{76}$
12	RK_2^{24}	2^{28}	$X_3^{24}, X_5^{24}, X_6^{26}, X_7^{26}, \boxed{X_{12}^{27}}, \boxed{X_{13}^{27}}, X_{15}^{24}$	$2^{40} \times 2^{32} = 2^{72}$
13	RK_6^{26}	2^{28}	$X_3^{24}, X_5^{24}, \boxed{X_6^{26}}, X_7^{26}, X_{12}^{26}, \boxed{X_{13}^{26}}, X_{15}^{24}$	$2^{44} \times 2^{28} = 2^{72}$
14	RK_4^{24}	2^{24}	$X_3^{24}, X_5^{24}, X_7^{26}, \boxed{X_9^{25}}, X_{12}^{26}, \boxed{X_{15}^{24}}$	$2^{48} \times 2^{28} = 2^{76}$
15	RK_6^{23}	2^{20}	$\boxed{X_3^{24}}, X_5^{24}, X_7^{26}, X_{12}^{26}, \boxed{X_{13}^{23}}$	$2^{48} \times 2^{24} = 2^{72}$
16	RK_4^{22}	2^{16}	$X_5^{24}, \boxed{X_7^{26}}, X_9^{22}, \boxed{X_{12}^{26}}$	$2^{48} \times 2^{20} = 2^{68}$
17	RK_2^{25}	2^{12}	$\boxed{X_5^{24}}, \boxed{X_5^{25}}, X_9^{22}$	$2^{52} \times 2^{16} = 2^{68}$
18	RK_0^{23}	2^8	$\boxed{X_1^{23}}, \boxed{X_9^{22}}$	$2^{56} \times 2^{12} = 2^{68}$
19	RK_5^{21}	2^4	$\boxed{X_{11}^{21}}$	$2^{56} \times 2^8 = 2^{64}$
20	RK_7^{20}	1	$\bigoplus Z_7^{20} = \bigoplus S(X_{11}^{21} \oplus RK_7^{20})$	$2^{56} \times 2^4 = 2^{60}$

Table 7.5: The data, time, and memory complexities using multiple data structures.

	Data	Time Complexity	Memory
1	2^{69}	$2^{69} + 1 \times \frac{2^{78.13}+2^{59.91}}{8\times26} + \frac{2^{76}}{8\times26} + \frac{145\times2^{72}}{8\times26} + 2^{76} + 2^{12} \approx 2^{76.11}$	$2^{66.04}$
2	2^{70}	$2^{70} + 2 \times \frac{2^{78.13}+2^{59.91}}{8\times26} + \frac{2^{76}+2^{72}}{8\times26} + \frac{145\times2^{68}}{8\times26} + 2^{72} + 2^{8} \approx 2^{73.03}$	$2^{67.04}$
3	$2^{70.58}$	$2^{70.58} + 3 \times \frac{2^{78.13}+2^{59.91}}{8\times26} + \frac{2^{76}+2^{72}+2^{68}}{8\times26} + \frac{145\times2^{64}}{8\times26} + 2^{68} + 2^{4} \approx 2^{72.62}$	$2^{67.62}$
4	2^{71}	$2^{71} + 4 \times \frac{2^{78.13}+2^{59.91}}{8\times26} + \frac{2^{76}+2^{72}+2^{68}+2^{64}}{8\times26} + \frac{145\times2^{60}}{8\times26} + 2^{64} \approx 2^{72.95}$	$2^{68.04}$

We then access the hash tables (H_Z, H_X) sequentially to retrieve 2 4-bit words. For simplicity, we consider the time to retrieve a single 4-bit word as a one F-function operation. Therefore, for this step, we need $2^{56} \times (1 + 2^{20}) \approx 2^{76}$ F-function operations. Consequently, we got 2^{72} 76-bit candidates of the round keys. As shown in Appendix A, we need 145 F-function operations for each candidate to get the corresponding 2^4 80-bit candidates of the master key. The exhaustive search over the candidates to get the right master key takes $2^{76} + 2^{12}$ 26-round encryptions. Therefore, the total time complexity is $2^{69} + 1 \times \frac{2^{78.13}+2^{59.91}}{8\times 26} + \frac{2^{76}}{8\times 26} + \frac{145\times 2^{72}}{8\times 26} + 2^{76} + 2^{12} \approx 2^{76.11}$ 26-round encryptions. The memory complexity is dominated by storing the part of the ciphertexts involved during the computation of $\bigoplus Z_7^{20}$ (the state $\mathcal{S}_0$) which is 2^{66} 66-bit blocks that is equivalent to $2^{66.04}$ 64-bit blocks. As shown in Table 7.5, the lowest time complexity can be achieved using 3 data structures and in this case the data, time, and memory complexities are $3 \times 2^6 \times 2^{63} = 2^{70.58}$ chosen tweak-plaintext combinations, $2^{72.62}$ 26-round encryprions, and $2^{67.62}$ 64-bit blocks, receptively.

7.4.2 Attack on 27-Round T-TWINE-128

The ciphertexts of 27-round of T-TWINE-128 can be written as X^{28}. The process of obtaining $\bigoplus X_{15}^{20}$ involves the following 35 round keys:

$$RK^{27}, RK^{26}, RK^{25}_{[0,1,2,3,4,5,7]}, RK^{24}_{[0,1,2,6,7]}, RK^{23}_{[0,4,6]}, RK^{22}_{[4,5]}, RK^{21}_{5}, RK^{20}_{7}$$

However, we only need to guess 116 bits in 29 round keys and the other 6 round keys can be computed based on the key schedule as follows:

$$\begin{aligned} RK_0^{23} &= RK_4^{27}, & RK_6^{23} &= RK_2^{27}, \\ RK_4^{22} &= RK_6^{27} \oplus S(RK_7^{27}), & RK_5^{22} &= RK_0^{26}, \\ RK_5^{21} &= RK_0^{25}, & RK_7^{20} &= RK_1^{27} \oplus S(RK_5^{25}) \oplus (0\|CON_L^{23}) \oplus (0\|CON_H^{26}). \end{aligned}$$

where CON_L^{23} and CON_H^{26} are predefined constants.

Key Recovery Procedure. Using the same procedure we have applied in the previous section, we can recover 2^{112} 116-bit candidates of the round keys and then retrieve 2^{124} 128-bit candidates of the master key by guessing 12 bits. The number of the candidates can be reduced by repeating the attack several times using different values of the constant bits in the data structure.

Attack Complexity. When we use a single data structure, we need approximately $2^{113.83}$ F-function operations to fill the hash tables, then we need additionally 2^{116} F-function operations to access the tables and recover 2^{112} 116-bit candidates. Thus, we retrieve the right master key using 2×2^{124} 27-round encryptions. By repeating the attack 6 times, we need $\frac{1}{8\times 27} \times (6 \times 2^{113.83} + 2^{116} + 2^{112} + \cdots + 2^{96}) + 2^{104} + 2^{40} = 2^{109.54}$ 27-round encryptions to retrieve the right master key. Hence, the data complexity is $6 \times 2^{69} = 2^{71.58}$ chosen tweak-plaintext combinations. The memory complexity is dominated by storing the values of $\bigoplus Z_7^{20}$ in the hash table H_Z. Therefore, we need 6×2^{92} 4-bit blocks which is equivalent to $2^{90.58}$ 64-bit blocks.

7.5 Attacking One More Round

Chu *et al.* [21] presented a general method to use the dynamically chosen plaintexts idea in order to attack one more round in the integral cryptanalysis by adding this round before the distinguisher. In general, appending rounds before the integral distinguisher may lead to use the full codebook of the plaintext. However, the dynamically chosen plaintext method guarantees that we will not use the full codebook of the plaintext. In this section, we explain how we can prepend one round before the integral distinguisher. Consequently, we can target 27 and 28 rounds of T-TWINE-80 and T-TWINE-128, respectively.

The core idea of the method is to express one of the constant bits (c) of the distinguisher input as a non-linear boolean function in some plaintext bits (x) and key bits (k), *i.e.*, $c = f(x, k)$. Then, we guess the key bits (k) and carefully select a specific plaintext set $\mathfrak{D}_k^c$ that guarantees the constant bit c is fixed to 0 or 1 while the other bits satisfy the distinguisher input. Consequently, the whole plaintext set used during the attack will be $\bigcup \mathfrak{D}_k^c$.

In our attack, the plaintext is X^1 and the distinguisher input is X^2. Therefore, we have to select the plaintexts such that $X_6^2[3]$ (the most significant bit of X_6^2) is fixed to 0 or 1 while the other bits of X^2 are active. From T-TWINE structure, $X_6^2[3] = X_9^1[3] \oplus S(X_8^1 \oplus k)[3]$ where $k = RK_4^1 \oplus RT_2^1$.

Based on the algebraic normal form of T-TWINE's S-box, $X_6^2[3]$ can be expressed as follows:

$$\begin{aligned} X_6^2[3] =& X_9^1[3] \oplus 1 \oplus x[0] \oplus x[2] \oplus (x[0] \cdot x[1]) \oplus (x[1] \cdot x[2]) \oplus (x[0] \cdot x[1] \cdot x[2]) \\ & \oplus (x[0] \cdot x[1] \cdot x[3]) \oplus (x[1] \cdot x[2] \cdot x[3]) \end{aligned}$$

where $x[i] = X_1^8[i] \oplus k[i]$ and $k[i] = RK_4^1[i] \oplus RT_2^1[i]$. Therefore, $X_6^2[3]$ depends on the 5 bits $X_8^1||X_9^1[3]$ and the 4 bits of the round key RK_4^1.

The procedure to determine the suitable plaintext set in our attack is as follows:

1. Initialize 32 empty lists namely $\mathfrak{D}_k^0$ and $\mathfrak{D}_k^1$ where $0 \leq k \leq 15$.

2. For each possible value of k and for all 2^5 possible values of $X_8^1||X_9^1[3]$, compute $X_6^2[3]$ and store $X_8^1||X_9^1[3]$ in $\mathfrak{D}_k^0$ if $X_6^2[3]$ is 0 or in $\mathfrak{D}_k^1$ if $X_6^2[3]$ is 1.

3. For each $C := \{c_k | 0 \leq k \leq 15\} \in \mathbb{F}_2^{16}$, if $|\bigcup_k \mathfrak{D}_k^{c_k}| < 2^5$, export C and its corresponding $\{\mathfrak{D}_k^{c_k}\}$ as a possible plaintext set.

Based on our evaluation, there are 32 plaintext sets of $\{\mathfrak{D}_k^{c_k}\}$. In each set, there are 31 out of 32 possible values of $X_8^1||X_9^1[3]$. To validate these sets, we perform an extra step as follows: for each k, we construct X_8^1, X_9^1 such that $X_8^1||X_9^1[3] \in \mathfrak{D}_k^{c_k}$ and $X_9^1[2]||X_9^1[1]||X_9^1[0]$ takes all possible values, then compute $X_6^2 = X_9^1 \oplus S(X_8^1 \oplus k)$, after that, we check if $X_8^1||X_6^2[2]||X_6^2[1]||X_6^2[0]$ takes all 128 possible values and $X_6^2[3] = c_k$ or

Table 7.6: An example of $\{\mathfrak{D}_k^{c_k}\}$

$RK_4^1 \oplus RT_2^1$	$\mathfrak{D}_k^{c_k} = \{X_8^1 \vert\vert X_9^1[3]\}$	$X_6^2[3]$
0000	00001, 00010, 00101, 00111, 01000, 01011, 01101, 01110, 10001, 10010, 10101, 10110, 11000, 11011, 11100, 11110	0
0001	00001, 00010, 00100, 00110, 01000, 01011, 01101, 01110, 10001, 10010, 10101, 10110, 11000, 11011, 11101, 11111	1
0010	00001, 00011, 00101, 00110, 01001, 01010, 01100, 01111, 10001, 10010, 10101, 10110, 11000, 11010, 11100, 11111	0
0011	00001, 00011, 00100, 00111, 01000, 01011, 01101, 01110, 10000, 10011, 10100, 10111, 11000, 11010, 11101, 11110	0
0100	00001, 00010, 00100, 00111, 01000, 01011, 01100, 01110, 10001, 10010, 10101, 10111, 11000, 11011, 11100, 11111	1
0101	00001, 00010, 00100, 00111, 01000, 01011, 01101, 01111, 10001, 10010, 10100, 10110, 11000, 11011, 11100, 11111	0
0110	00001, 00010, 00100, 00111, 01001, 01011, 01101, 01110, 10000, 10010, 10100, 10111, 11001, 11010, 11101, 11110	0
0111	00001, 00010, 00100, 00111, 01000, 01010, 01101, 01110, 10001, 10011, 10100, 10111, 11001, 11010, 11101, 11110	1
1000	00001, 00010, 00101, 00110, 01000, 01011, 01100, 01110, 10001, 10010, 10101, 10111, 11000, 11011, 11101, 11110	0
1001	00001, 00010, 00101, 00110, 01000, 01011, 01101, 01111, 10001, 10010, 10100, 10110, 11000, 11011, 11101, 11110	1
1010	00001, 00010, 00101, 00110, 01000, 01010, 01100, 01111, 10001, 10011, 10101, 10110, 11001, 11010, 11100, 11111	0
1011	00001, 00010, 00101, 00110, 01001, 01011, 01100, 01111, 10000, 10010, 10101, 10110, 11001, 11010, 11100, 11111	1
1100	00001, 00010, 00101, 00111, 01000, 01011, 01100, 01111, 10001, 10010, 10100, 10111, 11000, 11011, 11100, 11110	1
1101	00001, 00010, 00100, 00110, 01000, 01011, 01100, 01111, 10001, 10010, 10100, 10111, 11000, 11011, 11101, 11111	0
1110	00001, 00011, 00101, 00110, 01000, 01011, 01100, 01111, 10000, 10011, 10101, 10110, 11000, 11010, 11100, 11111	1
1111	00001, 00011, 00100, 00111, 01001, 01010, 01101, 01110, 10001, 10010, 10100, 10111, 11000, 11010, 11101, 11110	1

not. Table 7.6 depicts an example of these sets in which $X_8^1||X_9^1[3]$ does not take the value of 000000.

Data Collection. We firstly construct a data structure in which all bits of X^1 take all the possible values except $X_8^1||X_9^1[3] \in \bigcup \mathfrak{D}_k^{c_k}$. For the tweak, all bits are fixed to constant except the 6 bits (T_3, $T_{12}[0,2]$) take all the possible values. Then, we ask the encryption oracle to obtain the corresponding ciphertexts and store the ciphertext associated with the active bits of the tweak in a hash table indexed by the value of $X_8^1||X_9^1[3]$. Therefore, the data complexity of a single structure is $2^6 \times (2^{64} - 2^{59}) \approx 2^{69.95}$ chosen tweak-plaintext combinations.

7.5.1 Key Recovery Attacks

T-TWINE-80. We firstly guess the value of RK_4^1 and based on the value of $k = RK_4^1 \oplus RT_2^1$, we select a set of 2^{69} ciphertexts corresponding to the plaintexts that include $\mathfrak{D}_k^{c_k}$. After that, we apply the same steps described in Section 7.4.1 to obtain 2^{72} candidates of the 76-bit round keys. It should be mentioned that the *relative* relations between the round keys involved in the analysis rounds are the same as in Section 7.4.1.

Using each value of RK_4^1 combined with 2^{72} 76-bit candidates of the round keys, we can compute 2^{72} 80-bit candidates of the master key. Subsequently, we get in total $2^4 \times 2^{72} = 2^{76}$ 80-bit candidates of the master. The right master key can be retrieved by the exhaustive search over these candidates using 2 pairs of plaintext/ciphertext.

The time complexity is $2^{69.95} + 2^4 \times (\frac{2^{78.13}+2^{59.91}}{8\times 27} + \frac{2^{76}}{8\times 27} + \frac{145\times 2^{72}}{8\times 27}) + 2^{76} + 2^{12} \approx 2^{76.47}$ 27-round encryptions. The time complexity can be reduced to $2^{75.79}$ 27-round encryptions if we use two data structures ($2 \times 2^{69.95} = 2^{70.95}$ chosen tweak-plaintext combinations). The memory complexity is dominated by storing the ciphertexts associated with the active bits of the tweak. Therefore, the memory complexity will be $2^{71.08}$ 64-bit blocks.

T-TWINE-128. In the same manner, we can target 28 rounds of T-TWINE-128. By repeating the attack using different 5 data structures, we can retrieve the right master key. The data complexity is $5 \times 2^{69.95} = 2^{72.27}$ chosen tweak-plaintext combinations. The time complexity is $2^{113.38}$ 28-round encryptions. The memory complexity is $5 \times 2^4 \times 2^{92}$ 4-bit blocks which is equivalent to $2^{94.32}$ 64-bit blocks.

7.6 Summary

We studied the security of T-TWINE against integral cryptanalysis. In particular, we showed that adding a tweak to the round function structure gives the attacker more room to target a large number of rounds in T-TWINE compared to TWINE. More precisely, we are able to construct several integral distinguishers that cover 19 rounds of T-TWINE whereas the longest distinguisher covers only 16 rounds of TWINE. Furthermore, we launched key recovery attacks against 27 and 28 of T-TWINE-80 and T-TWINE-128, respectively. The presented attacks are the best-published attacks against both variants of T-TWINE.

Chapter 8

On MILP-based Automatic Search for Bit-based Division Property for Ciphers with (large) Linear Layers

With the introduction of the division trail, the bit-based division property (BDP) has become the most efficient method to search for integral distinguishers. The notation of the division trail allows us to automate the search process by modelling the propagation of the DBP as a set of constraints that can be solved using generic Mixed-integer linear programming (MILP) and SMT/SAT solvers. The current models for the basic operations and S-boxes are efficient and accurate. In contrast, the two approaches to model the propagation of the BDP for the non-bit-permutation linear layer are either inaccurate or inefficient. The first approach relies on decomposing the matrix multiplication of the linear layer into `COPY` and `XOR` operations. The model obtained by this approach is efficient, in terms of the number of the constraints, but it is not accurate and might add invalid division trails to the search space, which might lead to missing the balanced property of some bits. The second approach employs a one-to-one map between the valid division trails through the primitive matrix represented the linear layer and its invertible sub-matrices. Despite the fact that the current model obtained by this approach is accurate, it is inefficient, *i.e.,* it produces a large number of constraints for large linear layers like the one of Kuznyechik [86]. In this paper, we address this problem by utilizing the one-to-one map to propose a new MILP model and a search procedure for large non-bit-permutation layers. As a proof of the effectiveness of our approach, we improve the previous 3- and 4-round integral distinguishers of Kuznyechik [86] and the 4-round one of PHOTON's internal permutation (P_{288}) [47]. We also report, for the fist time, a 4-round integral distinguisher for Kalyna block cipher [74] and a 5-round integral distinguisher for PHOTON's internal permutation (P_{288}).

8.1 Introduction

The first search tool utilized the bit-based division property was limited to building integral distinguishers for block ciphers with block size less than 32 bits since the complexity of the search is around $\mathcal{O}(2^n)$ where n is the block size [98]. Xiang *et al.* [104] have overcome the problem of the restriction on the block size by proposing the *division trails.* Using the division trial, the search process for an integral distinguisher can be converted to checking whether a specific division trail exists or not. They also proposed a systematic method to model the propagation rules of the BDP as a set of linear constraints. Hence, the search process can be efficiently automated with the help of generic Mixed Integer Linear Programming (MILP) and SAT solvers. Moreover, Xiang *et al.* provided an accurate model for the propagation of the BDP through the basic operations; `COPY`, `XOR`, and `AND`, in addition to an accurate model for S-boxes. With the help of these models, it is now feasible to look for integral distinguishers for many ciphers that utilize these operations when the used linear layer is a bit-permutation.

For ciphers with non-bit-permutation linear layers, Sun *et al.* [90] proposed a model relying on decomposing the matrix corresponding to the linear layer into its basic operations; COPY and XOR. We refer to this model through our paper as *Disjointed Representation* and we will provide more details about it in the following sections. The main two advantages of this model are: (i) it is applicable to all kinds of linear layers, and (ii) the number of constraints needed to model the propagation of the BDP is small, precisely, $2n$ where n denotes the size of the matrix input in bits. However, this representation does not model the propagation accurately and might add invalid division trails to the search space which might lead to missing the balanced property of some bits.

Another model for the propagation of the BDP through non-bit-permutation linear layers is presented by Zhang and Rijmen in [106]. They observed that there is a one-to-one map between each valid division trail and one of the invertible sub-matrices of the matrix, M, representing the linear layer. They were able to convert this map to a set of MILP constraints. Unlike the first model provided by [90], the new model is more accurate. However, the number of the MILP constraints grows exponentially with the size of M. Recently, Hu *et al.* partially solved this problem in [50] by utilizing the one-to-one relation to build a model of 4-degree constraints that can be solved using SMT/SAT. The new number of the constraints is proportional to the square of matrix size. Unfortunately, this model is still not suitable for some large linear layers such as the one of Kuznyechik [86].

In this chapter, we propose a new model for the propagation of the BDP through large linear layers. In particular, we utilize the same one-to-one map proposed by Zhang and Rijmen to derive a set of constraints that filter out all non-invertible sub-matrices, part of them during the offline modelling process and the other part on-the-fly during the search process. In order to validate the correctness of our approach, we use our model to reproduce the results of the 4- and 5-round key-dependent integral distinguishers of AES reported in [50]. With the help of our model, we improved the previous 3- and 4-round integral distinguishers of Kuznyechik block cipher and the 4-round one of PHOTON's internal permutation (P_{288}). We also report, for the fist time, a 4-round integral distinguisher for Kalyna block cipher [74] and a 5-round integral distinguisher for

Table 8.1: Integral distinguishers for Kuznyechik, Kalyna and PHOTON.

Ciphers	#Rounds	$\log_2$(Data)	Reference
Kuznyechik	3	$116^\star$	[14]
	3	56	Section 8.5.1
	4	$127^\star$	[14]
	4	120	Section 8.5.1
Kalyna-128	$4^\dagger$	64	Section 8.5.2
	$4^\S$	96	Section 8.5.2
	$4^\ddagger$	62	Section 8.5.2
PHOTON (P_{288})	4	48	[90]
	4	40	Section 8.5.3
	5	280	Section 8.5.3

* Higher-order differential.
† Without pre-whitening operation.
§ With pre-whitening operation.
‡ A key-dependent distinguisher which depends on the 32 least significant bits of the pre-whitening key.

PHOTON's internal permutation (P_{288}) [47]. Table 8.1 summarizes our results. The rest of this chapter is organized as follows. In Section 8.2, we recall some relevant definitions. In Section 8.3, we revisit the previous MILP models for the linear layers. Next, we illustrate in details our new model and search approach in Section 8.4. In Section 8.5, we show some applications of the new model. Finally, the chapter is concluded in Section 8.6.

8.2 Preliminaries

We represent n-bit vectors using bold letters, *e.g.,* $\boldsymbol{u} \in \mathbb{F}_2^n$. The i-th element of $\boldsymbol{u}$ is expressed as u_i and the Hamming weight $hw(\boldsymbol{u})$ is calculated as $hw(\boldsymbol{u}) = \sum_{i=0}^{n-1} u_i$. For a matrix $M \in \mathbb{F}_2^{p\times q}$, we use the notation $M(i,j)$ to represent the element of M located at the i-th row and j-th column, $r_i = M(i,*)$ to represent the i-th row, and $c_j = M(*,j)$

to represent the j-th column of M. Given two q-bit and p-bit vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, we define $M_{\boldsymbol{v},\boldsymbol{u}} \in \mathbb{F}_2^{hw(\boldsymbol{v})\times hw(\boldsymbol{u})}$ as a sub-matrix of M as follows

$$M_{\boldsymbol{v},\boldsymbol{u}} = [M(i,j)],\ s.t.\ v_i = u_j = 1, \forall\ 0 \le i \le p-1,\ 0 \le j \le q-1$$

Given a q-bit vector $\boldsymbol{u}$, we define $M_{\boldsymbol{u}} \in \mathbb{F}_2^{p\times hw(\boldsymbol{u})}$ as a sub-matrix of M as follows

$$M_{\boldsymbol{u}} = [M(*,j)],\ s.t.\ u_j = 1, \forall\ 0 \le j \le q-1$$

Definition 8.1 (Binary Matrix [50]) *Suppose for a matrix $M^{'} \in \mathbb{F}_{2^m}^{s\times s}$, we represent the element $M^{'}(i,j)$ in $M^{'}$ as a polynomial in the extension field $\mathbb{F}_{2^m} \simeq \mathbb{F}[x]/(f)$, where f is the irreducible polynomial over $\mathbb{F}_2$ with degree m, then we call $M^{'}$ a binary matrix if all such polynomials in $M^{'}$ can only be 0 or 1. Otherwise, $M^{'}$ is called a non-binary matrix.*

8.3 Previous MILP-based Modelling for Linear Layers

The propagation of the bit-based division property through bit-permutation linear layers, *e.g.,* the linear layers of PRESENT [18]and GIFT [4] , can be easily modelled by rearranging the variables based on the permutation. In contrast, the non-bit-permutation linear layers, *e.g.,* the linear layers of AES and Kuznyechik [29], needs a more complex model.

In this section, we revisit the two methods used to model the propagation of the BDP through non-bit-permutation linear layers. These methods relay on representing the matrix multiplication in the linear layer at the bit level. Suppose the linear layer can be represented as a matrix multiplication over the field $\mathbb{F}_{2^m}$ using the matrix $M^{'} \in \mathbb{F}_{2^m}^{s\times s}$. Given the irreducible polynomial of the field $\mathbb{F}_{2^m}$, we can derive a unique equivalent matrix $M \in \mathbb{F}_2^{n\times n}$ called the primitive matrix at the bit level where $n = s \times m$.

8.3.1 Disjointed Representation

Since the primitive matrix M is presented at the bit level, *i.e.,* $M(i,j) \in \{0,1\}$, we can decompose the linear layer into its basic operations, *i.e.,* `AND` with 0 or 1 and `XOR` operations. Consequently, the propagation of the BDP can be easily modelled using the models of the basic operations [90].

Let $\boldsymbol{u} \xrightarrow{M} \boldsymbol{v}$ denote the division trail through the linear layer where $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}_2^n$. By defining a set of auxiliary binary variables $\boldsymbol{t} = \{t_{(i,j)}$ if $M(i,j) = 1, 0 \leq i,j \leq n-1\}$, we can model the propagation of the BDP at the bit level in two steps as follows:

- $(u_j) \xrightarrow{\texttt{COPY}} (t_{(0,j)}, t_{(1,j)}, \ldots, t_{(n-1,j)})$ where

$$-u_j + \sum_{\substack{i=0 \\ M(i,j)=1}}^{n-1} t_{(i,j)} = 0$$

- $(t_{(i,0)}, t_{(i,1)}, \ldots, t_{(i,n-1)}) \xrightarrow{\texttt{XOR}} (v_i)$ where

$$-v_i + \sum_{\substack{j=0 \\ M(i,j)=1}}^{n-1} t_{(i,j)} = 0$$

Hence, the total number of constraints $\#\mathcal{L} = 2n$.

Limitations. Despite the fact that this method is simple and efficient in terms of the number of constraints, it cannot handle the cancellation between monomials since it handles each output bit individually. Hence, it is not precise and it might produce invalid division trails leading to missing the balanced property of some output bits. For further details, see [106].

8.3.2 Compact Representation

One method to overcome the problem of the monomial cancellations is to deal with the linear layer as a one single block like an S-box. However, the large size of the linear layer renders this approach computationally infeasible in many cases.

In this context, Zhang and Rijmen observed that there is a one-to-one map between the accurate division trails of the primitive matrix M and invertible sub-matrices of M [106]. This observation is stated in the following theorem.

Theorem 8.1 ([**106**]) *Let M be the $n \times n$ primitive matrix of an invertible linear transformation and $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}_2^n$. Then $\boldsymbol{u} \xrightarrow{M} \boldsymbol{v}$ is one of the valid division trails of the linear transform M if and only if $M_{\boldsymbol{v},\boldsymbol{u}}$ is invertible.*

Using this one-to-one map, they proposed a systematic method to model a binary matrix $M' \in \mathbb{F}_{2^m}^{s \times s}$ as a set of MILP constraints. For more derails, see [106]. In this case, the total number of constraints $\#\mathcal{L} = m \times (2^s - 1)$.

Regarding the non-binary matrices, we can still use the same method, but the number of constrains will exponentially increase with the size of the primitive matrix, *i.e.,* if the primitive matrix M is $n \times n$, then the total number of constraints $\#\mathcal{L} = 2^n - 1$.

Hu *et al.* presented an updated version of Theorem 8.1 in [50]. They removed the restriction that the primitive matrix M must be invertible to have valid division trails. Consequently, the primitive matrix M could be in general of size $p \times q$. Hence, $\boldsymbol{u} \xrightarrow{M} \boldsymbol{v}$ is one of the valid division trails of M if and only if $M_{\boldsymbol{v},\boldsymbol{u}}$ is invertible where $\boldsymbol{u}$ and $\boldsymbol{v}$ are q- and p-bit vectors, and $hw(\boldsymbol{u}) = hw(\boldsymbol{v})$. Hu *et al.* also utilized this one-to-one map to present a new model for the propagation of the BDP through a non-binary matrix using less number of constraints. If a primitive matrix M is $p \times q$, then the total number of constraints will be $\#\mathcal{L} = p^2$. It should be mentioned that the constraints are 4-degree ones, therefore it is solvable using SMT/SAT solvers and cannot be handled using MILP solvers. For more details, see [50].

Limitations. Even though the models by Zhang and Rijmen, and Hu *et al.* are accurate, they are inefficient for large linear layers, *e.g.,* the primitive matrix corresponding to the linear layer of Kuznyechik is 128×128, therefore we will need 2^{128} or $128^2 = 2^{14}$ constraints to model a single linear layer if we use Zhang and Rijmen and Hu *et al.* methods receptively. Therefore, when the distinguisher covers many rounds, it will be computationally infeasible for current MILP/SAT solvers to handle the model due to the large number of the constraints.

8.4 MILP-based Modelling for (large) Linear Layers

As mentioned in the previous section, the current models for the non-bit-permutation linear layer in the literature are either inaccurate or inefficient for large linear layers. In this paper, we tackle this problem by proposing an accurate model for the linear layer when its input division property is priorly known before the modelling step. Thereby, this model is more suitable for the first round of the distinguisher. Regarding the other rounds of the distinguisher when the input division property cannot be determined during the modelling, we use the disjointed representation described in Section 8.3.1 and address its inaccuracy by discarding any invalid trails on-fly during the search process.

8.4.1 Prior-Known Input Division Property to the Linear Layer

Suppose the primitive matrix M is of size $p \times q$ and let $\boldsymbol{u}$ be the input division property to M and assume it is determined a priori. Consequently, we can utilize Theorem 8.1 and its updated version in [50] to derive all correct division trails. The naive method to do so is by exhaustively trying all the values of the output division property $\boldsymbol{v}$ such that $hw(\boldsymbol{u}) = hw(\boldsymbol{v})$ and checking if the sub-matrix $M_{\boldsymbol{v},\boldsymbol{u}}$ is invertible. Despite the correctness of this method, we need to try $\binom{p}{hw(\boldsymbol{u})}$ sub-matrices which is a very large number in almost all the cases. Moreover, we have to find a method to encode these division trails as MILP constraints to build a large model that covers many number of rounds. In the following, we explain our main idea to overcome this problem.

Main Idea. Based on Theorem 8.1, the sub-matrix $M_{\boldsymbol{v},\boldsymbol{u}}$ must be invertible to have a valid trail $\boldsymbol{u} \xrightarrow{M} \boldsymbol{v}$, *i.e.,* the sub-matrix $M_{\boldsymbol{v},\boldsymbol{u}}$ must not include linearly dependant rows. Given the input division property $\boldsymbol{u}$, we can construct the column matrix $M_{\boldsymbol{u}}$. Subsequently, we can get the row echelon form of $M_{\boldsymbol{u}}$ using the Gaussian eliminations, and obtain all the sets of linearly dependent rows. Then, instead of checking each value of $\boldsymbol{v}$ (as in the naive method), we derive a set of constraints that guarantee the bits v_i do not lead to including any set of linearly dependent rows from $M_{\boldsymbol{u}}$. In order to complete the model, one more constraint should be added to enforce $hw(\boldsymbol{u}) = hw(\boldsymbol{v})$. Hence, the value of $\boldsymbol{v}$ that satisfies these constraints is indeed a valid output division property.

The following examples illustrates our idea.

Detailed Example. Assume a toy linear layer where its primitive matrix M is 8×8. Given the input division property $\boldsymbol{u} = (1, 1, 1, 1, 1, 0, 0, 0)$, we can construct the column matrix $M_{\boldsymbol{u}}$ by choosing the columns of M that correspond to the nonzero bits in $\boldsymbol{u}$.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \xrightarrow{\boldsymbol{u}} M_{\boldsymbol{u}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

We follow the procedure given below to derive a set of linear constraints as a function in the output division property $\boldsymbol{v} = (v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7)$ to trace the propagation of the division property for $M_{\boldsymbol{u}}$.

1. Check whether $Rank(M_{\boldsymbol{u}}) = hw(\boldsymbol{u})$ to ensure that there is at least one full rank (invertible) sub-matrix, and hence at least one valid division trail. Otherwise, we conclude that $\boldsymbol{u}$ cannot be propagated to any valid $\boldsymbol{v}$.

2. Use Gaussian eliminations to put $M_{\boldsymbol{u}}$ in its row echelon form while keeping track the row operations. Hence, each all-zero row in the row echelon form implies a set of linearly dependent rows in the original matrix $M_{\boldsymbol{u}}$, *e.g.,* the first all-zero row in our example can be expressed as $r_0 + r_1 + r_5 = \mathbf{0}$ which means that the rows $\{r_0, r_1, r_5\}$ from $M_{\boldsymbol{u}}$ are linearly dependent. The details of the Gaussian eliminations for our example can be found in Appendix B.

$$\left[\begin{array}{ccccc|c} 1&0&0&0&0&r_0\\ 1&1&0&0&1&r_1\\ 0&1&0&0&0&r_2\\ 0&0&1&0&0&r_3\\ 0&0&0&1&0&r_4\\ 0&1&0&0&1&r_5\\ 1&1&0&0&0&r_6\\ 0&0&1&1&0&r_7 \end{array}\right] \xrightarrow[\textit{Elimination}]{\textit{Gaussian}} \left[\begin{array}{ccccc|l} 1&0&0&0&0&r_0\\ 0&1&0&0&1&r_1+r_0\\ 0&0&1&0&0&r_3\\ 0&0&0&1&0&r_4\\ 0&0&0&0&1&r_2+r_1+r_0\\ 0&0&0&0&0&r_5+r_1+r_0\\ 0&0&0&0&0&r_6+r_2+r_0\\ 0&0&0&0&0&r_7+r_3+r_4 \end{array}\right] \rightarrow \left\{\begin{array}{l} r_0+r_1+r_5=\mathbf{0}\\ r_0+r_2+r_6=\mathbf{0}\\ r_3+r_4+r_7=\mathbf{0} \end{array}\right\}$$

In general, if $M_{\boldsymbol{u}}$ is $p \times hw(\boldsymbol{u})$, then there are $p - hw(\boldsymbol{u})$ all-zero rows in the row echelon form given that $Rank(M_{\boldsymbol{u}}) = hw(\boldsymbol{u})$.

3. Find all the sets of linearly dependent rows. We do so by trying the combinations between the relations derived from all-zero rows obtained in the previous step, *e.g.,* combine $r_0+r_1+r_5 = \mathbf{0}$ and $r_0+r_2+r_6 = \mathbf{0}$ will produce $r_0+r_1+r_5+r_0+r_2+r_6 =$

$\mathbf{0} \Rightarrow r_1 + r_2 + r_5 + r_6 = \mathbf{0}$ which means the rows $\{r_1, r_2, r_5, r_6\}$ are linearly dependent.

$$\begin{cases} r_0 + r_1 + r_5 = \mathbf{0} \\ r_0 + r_2 + r_6 = \mathbf{0} \\ r_3 + r_4 + r_7 = \mathbf{0} \\ r_1 + r_2 + r_5 + r_6 = \mathbf{0} \\ r_0 + r_1 + r_3 + r_4 + r_5 + r_7 = \mathbf{0} \\ r_0 + r_2 + r_3 + r_4 + r_6 + r_7 = \mathbf{0} \\ r_1 + r_2 + r_3 + r_4 + r_5 + r_6 + r_7 = \mathbf{0} \end{cases}$$

4. For each set of linearly dependent rows, we derive a constraint on some bits of $\boldsymbol{v}$ enforcing any selected sub-matrix to be invertible, *e.g.,* $r_0 + r_1 + r_5 = \mathbf{0}$ means the rows $\{r_0, r_1, r_5\}$ are linearly dependent. In other words, these rows together must not be a part of any sub-matrix in order to have valid trails. Reflecting on $\boldsymbol{v}$, this means the bits v_0, v_1, v_5 cannot be 1 at the same time. We can represent this relation as a linear constrain $v_0 + v_1 + v_5 \leq 2$. The initial model for our toy linear layer includes:

$$\begin{cases} v_0 + v_1 + v_5 \leq 2 & \text{(C1)} \\ v_0 + v_2 + v_6 \leq 2 & \text{(C2)} \\ v_3 + v_4 + v_7 \leq 2 & \text{(C3)} \\ v_1 + v_2 + v_5 + v_6 \leq 3 & \text{(C4)} \\ v_0 + v_1 + v_3 + v_4 + v_5 + v_7 \leq 5 & \text{(C5)} \\ v_0 + v_2 + v_3 + v_4 + v_6 + v_7 \leq 5 & \text{(C6)} \\ v_1 + v_2 + v_3 + v_4 + v_5 + v_6 + v_7 \leq 6 & \text{(C7)} \\ v_0, \ldots, v_7 \text{ are binary variables} \end{cases}$$

5. Remove the redundancy constraints, *e.g.,* the constraint C5 is redundant because if

the constraints C1 and C3 are satisfied, then the constraint C5 is satisfied. Also, if the constraints C1 and C3 are not satisfied, then the constraint C5 is not satisfied. In contrast, if one of the constraints C1 and C3 is satisfied and the other is not satisfied, the solution will be rejected even though the constraint C5 is satisfied. We can automate this step by checking if a set of dependent rows (A) is a sub-set of another set of dependent rows (B), then the constraint on the set B is redundant. The model for our toy linear layer is then reduced to:

$$\begin{cases} v_0 + v_1 + v_5 \leq 2 \\ v_0 + v_2 + v_6 \leq 2 \\ v_3 + v_4 + v_7 \leq 2 \\ v_1 + v_2 + v_5 + v_6 \leq 3 \\ v_0, \dots, v_7 \text{ are binary variables} \end{cases}$$

6. Finally, add a constraint to enforce that $hw(\boldsymbol{u}) = hw(\boldsymbol{v})$. The model for our toy linear layer will be

$$\begin{cases} v_0 + v_1 + v_5 \leq 2 \\ v_0 + v_2 + v_6 \leq 2 \\ v_3 + v_4 + v_7 \leq 2 \\ v_1 + v_2 + v_5 + v_6 \leq 3 \\ v_0 + v_1 + \cdots + v_7 = 5 \\ v_0, \dots, v_7 \text{ are binary variables} \end{cases}$$

Number of Constraints. Although we cannot count exactly the number of the required constraints before performing the procedure, we can give the upper bound of the number based on Step 3 as follows:

$$\#\mathcal{L} \leq 1 + \sum_{i=1}^{p-hw(\boldsymbol{u})} \binom{p - hw(\boldsymbol{u})}{i} = 1 + 2^{p-hw(\boldsymbol{u})} - 1 = 2^{p-hw(\boldsymbol{u})}$$

In the light of this upper bound, it is clear that the model is practically more applicable when $p - hw(\boldsymbol{u})$ is relatively small which is usually the case for the linear layer at the first round when we search for a distinguisher that covers a large number of rounds where the Hamming weight of the input division property of the distinguisher (the number of active bits) is very close to the block size.

8.4.2 Complete Model and Search Approach

In the previous section, we presented a model for the linear layer at the first round when its input division property is known before the modelling step. In this section, we propose a search approach allowing us to use that model even though the targeted distinguisher does not start from a linear layer. We also complete the model for the targeted distinguisher by showing how to handle the intermediate linear layers.

Intermediate Linear Layers. We use the disjointed representation described in Section 8.3.1 to model the intermediate linear layers. When a candidate division tail is obtained by solving the complete model, we then extract the values of the input and the output division property of each matrix multiplication in the trail. After that, we check whether $M_{\boldsymbol{v},\boldsymbol{u}}$ is invertible or not for each matrix multiplications. If one of them is not invertible, we discard the trail by updating the model through adding a special craft constraint and resolving the updated model.

Discarding Invalid Trails. Let $(u_0, \ldots, u_{n-1})$ and $(v_0, \ldots, v_{n-1})$ be the variables in the model representing the input and the output division property of a matrix multiplication where $M_{\boldsymbol{v},\boldsymbol{u}}$ is not invertible in the current solution of the model. Let I_0^u (I_1^u) be the indices of $\boldsymbol{u}$'s variables that equal to `0` (`1`) in the current solution. Similarly, let I_0^v (I_1^v) be the indices of $\boldsymbol{v}$'s variables that equal to `0` (`1`) in the current solution. We update the model based on the current solution by adding the following constraint

$$\sum_{i \in I_0^u}(u_i) + \sum_{i \in I_1^u}(1 - u_i) + \sum_{i \in I_0^v}(v_i) + \sum_{i \in I_1^v}(1 - v_i) \geq 1$$

Therefore, when we attempt to resolve the updated model, the current solution, *i.e.,* the invalid trial, will violate the new constraint and the solver will not consider it as a solution and try to obtain another solution.

Implementation. Although the models for both the first linear layer with known input division property and the intermediate linear layers with the discarding approach above are applicable using MILP and SMT/SAT, the approach to discard invalid trails is more efficient using MILP solvers via the callback function and the concept of lazy constraints [48, 51] without needing to resolve the model from scratch.

Last Linear Layer. When the distinguisher ends with a linear layer, we can model it using the disjointed representation (like the intermediate linear layers) or we can efficiently model it using the model for XOR operation. Let $(u_0, \ldots, u_{n-1})$ and $(v_0, \ldots, v_{n-1})$ be the variables in the model which represent the input and the output division property of the matrix multiplication in the last linear layer. Suppose we check if there is a division trail from the input division property of the distinguisher to the unit vector e_i, *i.e.,* checking if the i-th bit of the output is balanced or not. Therefore, the variables that represent the output division property will be set to

$$\begin{cases} v_i = 1 \\ v_l = 0, & 0 \leq l \leq n-1, l \neq i \end{cases}$$

Consequently, during modelling, we focus on row $r_i = M(i,*)$ of the primitive matrix M and the constraints on the input division property of the matrix multiplication will be

$$\begin{cases} \sum\limits_{\substack{j=0 \\ M(i,j)=1}}^{n-1} u_j = 1 \\ u_j = 0, \qquad 0 \leq j \leq n-1, M(i,j) = 0 \end{cases}$$

After solving the model, if there is a division trail from the input division property of the distinguisher to the unit vector e_i, we conclude that there are other division trails from the same input division property of the distinguisher to other unit vectors without creating/solving their corresponding models. The original division trial can be split into two sub-trails; from the input division property of the distinguisher to the input division property of the last linear layer $\boldsymbol{u}$, and from $\boldsymbol{u}$ to the unit vector e_i where $hw(\boldsymbol{u}) = hw(\boldsymbol{v}) = 1$, *i.e.,* only one variable from $(u_0, \ldots, u_{n-1})$ is 1 and the other are 0. Suppose this variable is u_j. Therefore, the column matrix $M_{\boldsymbol{u}}$ can be created from a single column $c_j = M(*,j)$. Based on Theorem 8.1, the division trail from the input division property of the distinguisher to the unit vector e_l, passing through $\boldsymbol{u}$, exists for the l-th output bit if $M(l,j) = 1$ where $0 \leq l \leq n-1$.

Search Approach. If the targeted distinguisher starts from a linear layer, the input division property of this linear layer is known and we can use the model described in Section 8.4.1. Hence, we create only one model for the distinguisher. Otherwise, we perform the following search approach:

1. We firstly determine all the possible values of the input division property of the first linear layer by propagating the input division property of the distinguisher through other parts of the first round, which is usually a non-linear layer of S-boxes.

2. Then, we check the i-th output bit by creating a group of sub-models starting from the first linear layer with different input division property, thereby, we can employ

the model described in Section 8.4.1 for the first linear layer in each sub-model.

3. Finally, we solve the sub-models independently in parallel by dividing our computational power between them. If the valid division trail that ends at the unity vector e_i exists for a sub-model, we terminate the search process for the other sub-models. If it does not exist for all sub-models, then the i-th output bit is balanced. The last two steps are repeated for all output bits.

Remark. Even though the model for the linear layer using the disjointed representation with discarding invalid trails approach is applicable to the first linear layer, we believe that modelling the first linear layer accurately from the beginning is important. Our reasoning for that is as follows. First, the Hamming weight of the input/output division property for the first linear layer is the highest compared to the successive linear layers, *i.e.,* the number of its possible propagation is high and the chance to find invalid sub-trails will increase, which leads to the second reason. Since every sub-trail in early rounds is branched to many trails in the successive rounds, invalid sub-trails in the first round have a larger effect on expanding the search space, and hence increasing the time of solving the model. We verified our hypothesis experimentally by comparing the running time to find the 4-round key-dependent integral distinguish of AES reported in [50] using the same platform in the two cases; the case when the first linear layer is modelled accurately from the beginning and the other case when we model the first linear layer using the disjointed representation with discarding approach. In the first case, the solver found the distinguisher in around 50 minutes. In contrast, the solver did not finish in the second case even after running for more than a day.

8.5 Applications of our New Approach

In this section, we report our findings when applying our approach to Kuznyechik and Kalyna block ciphers and a variant of PHOTON permutations. We also have reproduced the results of the 4- and 5-round dependent-key integral distinguishers of AES reported in [50].

During our experiments, We use either Gurobi* solver [48] or the CPLEX optimizer [51] to solve the models. Our source codes are available at `https://github.com/SubmissionAnonymou/MILP_DivisionProprerty_LinearLayer`

We use the following notation to present the integral property of each byte in the plaintext and ciphertext:

- $\mathcal{C}$: Each bit of the byte at the plaintext is fixed to constant.
- $\mathcal{A}$: All bits of the byte at the plaintext are active.
- $\mathcal{B}$: Each bit of the byte at the ciphertext is balanced (the XOR sum is zero).
- $\mathcal{U}$: A byte at the ciphertext with unknown status (the XOR sum is unknown).

When each bit of a byte has a different property, we use lowercase letters to present the property, *i.e.,* `c`, `a` and `b` will represent a constant bit, an active bit, and a balanced bit, respectively. For example, `caaaaaaa` represents a byte where the most significant bit is constant and the other bits are active.

In general during our experiments, when an R-round distinguisher is found, we follow two different paths in parallel as a next step; we examine whether $(R+1)$-round exists or not, and we try to find another R-round distinguisher that needs a less number of active bits, *i.e.,* less data complexity.

*We use the version of Groubi that has some problems reported in [36]. Therefore, when we find some balanced bits by solving a model using Gurobi and we could not verify this results by propagating the traditional integral property, we resolve the model again using the CPLEX optimizer in order to validate the results.

8.5.1 Application to Kuznyechik

The Russian encryption standard — Kuznyechik [29, 86], also known as GOST 34.12-2015, is a 9-round SPN-based block cipher with a 128-bit block size and 256 bits of key. The encryption procedure is performed as follows. After loading a block of 128-bit plaintext to a 16-byte internal state $\boldsymbol{x} = (x_0, \ldots, x_{15})$ where x_0 is the least significant byte, the state is Xored with a whitening round key (XOR Layer (X)). Then, the state is updated 9 times using an identical round function denoted as $R = (X \circ L \circ S)$ that consists of:

- Non-linear Layer (S): Each byte of the state is mapped using 8-bit S-box.
- Linear Layer (L): The 16-byte state is multiplied by 16×16 MDS matrix over the field $\mathbb{F}_{2^8}$ with the irreducible polynomial $X^8 + X^7 + X^6 + X + 1$.
- XOR Layer (X): The 16-bye state is Xored with the corresponding round key.

In [14], Biryukov *et al.* studied Kuznyechik security against the multiset-algebraic cryptanalysis in which they reported the 3- and 4-round integral distinguisher based on their algebraic degree.

3-round Integral Distinguishers. Biryukov *et al.* reported that the 3-round has degree at most 116 [14]. Therefore the XOR sum over a set of plaintexts with dimension 117 will be zero, *i.e.,* the 3-round integral distinguisher exists with the data complexity of 2^{117}. However, we found several 3-round integral distingushers with a much lower data complexity of 2^{56}. One of these distinguishers is as follows.

$$(\mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{C}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A})$$
$$\Downarrow 3R \circ X$$
$$(\mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B})$$

4-round Integral Distinguishers. Biryukov *et al.* also reported a 4-round distinguisher with the data complexity of 2^{127} depending on the 4-round has degree at most 126 [14]. In our experiments, we were able to find several 4-round integral distinguishers with data complexity of 2^{120} (120 active bits). One of these distingushers is as follows.

$$(\mathcal{C}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A})$$
$$\Downarrow 4R \circ X$$
$$(\mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B})$$

Other Experiments. Biryukov *et al.* extended the 4-round key-independent integral distinguisher to a 5-round key-dependent one with the same data complexity by appending the linear layer (L) before the 4-round one. The new distinguisher depends on the least significant byte of the master key. We were able to verify the existence of this distinguisher using our model by setting one bit to a constant and the other bits to active as shown below.

$$(\texttt{caaaaaaa}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A}, \mathcal{A})$$
$$\Downarrow 4R \circ X \circ L$$
$$(\mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B}, \mathcal{B})$$

As a next step, we employ the search approach proposed in the previous section to check the existence of the 5-round key-independent distinguisher with a single bit constant and 127 bits active, and we confirmed that this distinguisher does not exist even with the use of the accurate propagation of the BDP.

8.5.2 Application to Kalyna

The Ukrainian standard Kalyna [74], also known as DSTU 7624:2014, is a family of five SPN-based block ciphers denoted as Kalyna-l/k where $l, k \in \{128, 256, 512\}$ are the block size and the key size, respectively, such that $k = l$ or $k = 2 \times l$. The number of rounds depends on the key size.

We targeted the two members with the block size of 128 bits, Kalyna-128. The encryption procedure is performed as follows. The 16 bytes of the plaintext block $\boldsymbol{x} = (x_0, \ldots, x_{15})$ where x_0 is the least significant byte, is loaded to the 8×2 16-byte state matrix in column-wise order. After that, pre-whitening round key is added to each column independently using addition modulo 2^{64}. We denote this operation as $(\boxplus_{64})$. Then, The following round function denoted as $R = (X \circ L \circ SR \circ S)$ is iterated 10 or 14 times depending on the key size:

- Non-linear Layer (S): 4 different 8-bit S-boxes $\pi_s, s \in \{0, 1, 2, 3\}$ are used to map the bytes of the state matrix where the i-th byte (x_i) is substituted by $\pi_{i \bmod 4}(x_i)$.
- ShiftRows (SR): The bytes of each row in the state matrix are cyclically shifted to right by $\lfloor \frac{i}{4} \rfloor$ where $i, 0 \leq i \leq 7$ is the row index.
- Linear Layer (L): Each 8-byte column of the state matrix is independently multiplied by 8×8 MDS matrix over the field $\mathbb{F}_{2^8}$ with the irreducible polynomial $X^8 + X^4 + X^3 + X^2 + 1$.
- XOR Layer (X): the state matrix is Xored with the corresponding round key.

In the last round, the XOR Layer (X) is replaced by a post-whitening modular key addition modulo 2^{64}.

4-round Integral Distinguishers without pre-whitening. During our experiments, we found two 4-round integral distinguisher starting after the pre-whitening step with 8 active bytes as depicted below. The correctness of these distinguishers can be easily verified by propagating the integral properties though the equivalent structure of the round function. Given that, each 8-bit S-box is reused every 4 bytes and the first (second) 4 rows of the state matrix is shifted by the same step, the state matrix can be reconstructed as 2×2 matrix such that each 4 successive bytes are concatenated in a 32-bit word and the

4 different 8-bit S-boxes build a 32-bit super S-box. Therefore, when the diagonal (anti-diagonal) words of the new state matrix are active, *i.e.,* take all possible values from $\mathbb{F}_{2^{32}}^2$, the output after 4-rounds will be balanced similar to the 4-round integral distinguisher of AES [57].

$$\begin{bmatrix} \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \end{bmatrix} \text{OR} \begin{bmatrix} \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \end{bmatrix} \xRightarrow{4R} \begin{bmatrix} \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \end{bmatrix} \xrightarrow{Appending\ \boxplus_{64}} \begin{bmatrix} \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} \end{bmatrix} \text{OR} \begin{bmatrix} \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} \end{bmatrix} \xRightarrow{4R \circ \boxplus_{64}} \begin{bmatrix} \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \end{bmatrix}$$

4-round Integral Distinguishers with pre-whitening. We were able to extend each of the previous 4-round distinguishers to cover the pre-whitening operation. The new distinguishers need 12 active bytes as depicted above. In the following, we illustrate the way we use to select a set of plaintexts so that it satisfies the input division property of the 4-round distinguisher after applying the pre-whitening operation.

Since the pre-whitening operation is performed per column, we focus on each column independently. Suppose X, Y, and K denote a 64-bit word of the input, the output and the whitening key, respectively, such that $Y = X \boxplus_{64} K$. Each 64-bit word can be considered as the concatenation of two 32-bit words, *i.e.,* $X = X_l||X_r$, $Y = Y_l||Y_r$, and $K = K_l||K_r$. Therefore, $Y_r = X_r \boxplus_{32} K_r$ and $Y_l = X_l \boxplus_{32} K_l \boxplus_{32} C$ where $\boxplus_{32}$ denotes the addition modulo 2^{32} and C is the carry from the first addition part.

Consequently, a set of plaintexts such that X_r is fixed to constant and the 4 bytes of X_l takes all the possible values from $\mathbb{F}_{2^8}^4$, will give an output set such that Y_r will be constant and the 4 bytes of Y_r will take all the possible values from $\mathbb{F}_{2^8}^4$. This is because the whitening key is constant and the carry will be fixed over all the set's elements based on the previous two questions. As the result, we can easily satisfy one of the two column in the 4-round distinguishers.

The same method cannot be applied to the other column because if X_r takes all the possible values, Y_r will take all the possible values, but, the value of the carry will change depending on the value of the whitening key. Hence, we cannot adapt the values of X_l to enforce Y_l to be fixed over the set. To overcome this problem, we construct a set of plaintexts such that the 8 bytes of X take all the possible values from $\mathbb{F}_{2^8}^8$, hence, the 8 bytes of Y will take all the possible values from $\mathbb{F}_{2^8}^8$. As the result, the output set Y can be considered as 2^{32} sub-sets in which each sub-set satisfies the input division property of the other column of the 4-round distinguisher. Combining these two approaches, the 4-round distinguishers with the pre-whitening need 12 active bytes.

Using the BDP, we are able to verify the existence of these distingushers with the help of the propagation model of the BDP through modular addition with a constant presented in Chapter 6 (Section 6.2.1). Additionally, we have tried to reduce the number of active bits by iterating over the active bits one-by-one and set it to constant then check if the distinguisher still exists. Unfortunately, the distinguisher does not exist.

Other Experiments. During our experiments, we build a 4-round key-dependent distinguisher using 62 active bits. The new distinguisher depends on the 32 least significant bits of the pre-whitening key. The distinguisher starts at the linear layer of the first round with the input division property as shown below.

$$\begin{bmatrix} \texttt{ccaaaaaa} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \\ \mathcal{C} & \mathcal{A} \end{bmatrix} \xRightarrow{3R \circ X \circ L \circ SR} \begin{bmatrix} \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} \end{bmatrix}$$

8.5.3 Application to PHOTON

PHOTON [47] is a family of lightweight hash functions proposed by Guo *et al.* at CRYPTO 2011 and it has been standardized in ISO/IEC 29192-5:2016. PHOTON has 5 variants with 5 internal unkeyed permutations denoted as P_t where $t \in \{100, 144, 196, 256, 288\}$ is the internal state size. We target here the internal permutation P_{288}. The structure of the internal permutation follows the structure of AES where the internal state is represented as a $d \times d$ square matrix of cells. Thus, the internal state of P_{288} is a 6×6 matrix of bytes. Its round function consists of:

- AddConstants (X): Each byte of the 1st column of the state matrix is Xored with a round-dependent constant.
- SubCells (S): Each byte (x_i) of the state is substituted by $Sbox(x_i)$ where $Sbox$ is the 8-bit S-box of AES.
- ShiftRows (SR): The bytes of each row in state are cyclically shifted to left by i where $i \in 0 \leq i \leq 5$ is the row index.
- MixColumnsSerial (L): Each column of the state is independently multiplied by 6×6 MDS matrix over $\mathbb{F}_{2^8}$ with the irreducible polynomial $X^8 + X^4 + X^3 + X + 1$.

3- and 4-round Integral Distinguishers. Since the permutation is followed the AES structure, there are 3- and 4-round distinguishers that exploit the structure itself and independent on the used S-boxes and the MDS matrix. In particular, when the state matrix has a single byte active and the other bytes are constant (the data complexity is 2^8), each output bit after 3 rounds will have zero-sum (balanced). Also, there is a 4-round distinguisher when all diagonal's bytes of the state matrix are active (the data complexity is 2^{48}). In [90], Sun *et al.* verified the existence of these 3- and 4-round distinguishers using the MILP models for the propagation of the BDP. They have modelled the linear layer using the disjointed representation.

New 4-round Integral Distinguisher. At Crypto 2016, Sun *et al.* exploited a specific property of the matrix used in AES to introduce the first 5-round key-dependent integral distinguisher [87]. This property is that each column of the matrix has two equal elements. We employ a similar property to reduce the date complexity of the 4-round distinguisher of P_{288} and build a new 5-round one.

Suppose M_P and M_P^{-1} denote the matrix and its inverse that are used in P_{288} where

$$M_P = \begin{bmatrix} \mathtt{02} & \mathtt{03} & \mathtt{01} & \mathtt{02} & \mathtt{01} & \mathtt{04} \\ \mathtt{08} & \mathtt{0e} & \mathtt{07} & \mathtt{09} & \mathtt{06} & \mathtt{11} \\ \mathtt{22} & \mathtt{3b} & \mathtt{1f} & \mathtt{25} & \mathtt{18} & \mathtt{42} \\ \mathtt{84} & \mathtt{e4} & \mathtt{79} & \mathtt{9b} & \mathtt{67} & \mathtt{0b} \\ \mathtt{16} & \mathtt{99} & \mathtt{ef} & \mathtt{6f} & \mathtt{90} & \mathtt{4b} \\ \mathtt{96} & \mathtt{cb} & \mathtt{d2} & \mathtt{79} & \mathtt{24} & \mathtt{a7} \end{bmatrix}, \quad M_P^{-1} = \begin{bmatrix} \mathtt{15} & \mathtt{50} & \mathtt{eb} & \mathtt{62} & \mathtt{79} & \mathtt{99} \\ \mathtt{29} & \mathtt{a5} & \mathtt{c9} & \mathtt{c2} & \mathtt{fb} & \mathtt{2b} \\ \mathtt{56} & \mathtt{54} & \mathtt{8e} & \mathtt{9f} & \mathtt{e9} & \mathtt{57} \\ \mathtt{ae} & \mathtt{af} & \mathtt{03} & \mathtt{20} & \mathtt{c8} & \mathtt{ae} \\ \mathtt{47} & \mathtt{47} & \mathtt{01} & \mathtt{44} & \mathtt{8e} & \mathtt{46} \\ \mathtt{8c} & \mathtt{8d} & \mathtt{01} & \mathtt{8d} & \mathtt{02} & \mathtt{8d} \end{bmatrix}$$

Suppose $\boldsymbol{x} = (x_0, x_1, x_2, x_3, x_4, x_4)^T$ and $\boldsymbol{y} = (y_0, y_1, y_2, y_3, y_4, y_5)^T$ be the input and the output vectors to the matrix M_P such that $\boldsymbol{y} = M_P \times \boldsymbol{x}$. Suppose $\boldsymbol{x}$ take $2^{5\times 8=40}$ values where each of x_0, x_1, x_2, x_3 and x_4 take all the possible values from $\mathbb{F}_{2^8}$. Therefore, $\boldsymbol{y}$ will take 2^{40} values. Also, $\boldsymbol{x} = M_P^{-1} \times \boldsymbol{y}$ can be expressed as shown below

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_4 \end{bmatrix} = \begin{bmatrix} \mathtt{15} & \mathtt{50} & \mathtt{eb} & \mathtt{62} & \mathtt{79} & \mathtt{99} \\ \mathtt{29} & \mathtt{a5} & \mathtt{c9} & \mathtt{c2} & \mathtt{fb} & \mathtt{2b} \\ \mathtt{56} & \mathtt{54} & \mathtt{8e} & \mathtt{9f} & \mathtt{e9} & \mathtt{57} \\ \mathtt{ae} & \mathtt{af} & \mathtt{03} & \mathtt{20} & \mathtt{c8} & \mathtt{ae} \\ \mathtt{47} & \mathtt{47} & \mathtt{01} & \mathtt{44} & \mathtt{8e} & \mathtt{46} \\ \mathtt{8c} & \mathtt{8d} & \mathtt{01} & \mathtt{8d} & \mathtt{02} & \mathtt{8d} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix}$$

Hence, we can express x_4 as follows in equations (8.1) and (8.2).

$$x_4 = \mathtt{47} \cdot y_0 \oplus \mathtt{47} \cdot y_1 \oplus \mathtt{01} \cdot y_2 \oplus \mathtt{44} \cdot y_3 \oplus \mathtt{8e} \cdot y_4 \oplus \mathtt{46} \cdot y_5 \tag{8.1}$$

$$x_4 = \mathtt{8c} \cdot y_0 \oplus \mathtt{8d} \cdot y_1 \oplus \mathtt{01} \cdot y_2 \oplus \mathtt{8d} \cdot y_3 \oplus \mathtt{02} \cdot y_4 \oplus \mathtt{8d} \cdot y_5 \tag{8.2}$$

$$\mathtt{00} = \mathtt{cb} \cdot y_0 \oplus \mathtt{ca} \cdot y_1 \oplus \mathtt{00} \cdot y_2 \oplus \mathtt{c9} \cdot y_3 \oplus \mathtt{8c} \cdot y_4 \oplus \mathtt{cb} \cdot y_5 \tag{8.3}$$

From (8.1) and (8.2), we can derive the equation (8.3) which implies that $\{y_0, y_1, y_3, y_4, y_5\}$ are linearly dependent, *i.e.,* they can take at most $2^{4\times 8=32}$ values. Since $\boldsymbol{y}$ takes 2^{40} values, y_2 must take 2^8 values, *i.e.,* y_2 is an active bye and takes its all possible values($\mathcal{A}$).

Constructing 4-round Integral Distinguisher. We construct a set of 2^{40} chosen plaintexts such that the state matrix is as follows. The first 4 elements of the diagonal are active, the last two elements of the diagonal are equal and active (denoted as $\bar{\mathcal{A}}$), and the other elements of the state matrix are fixed to constant as shown below. After applying the three operations: AddConstants (X), SubCells (S), and ShiftRows (SR), the first column of the state matrix will be in the form of the vector $\boldsymbol{x}$. Therefore, the output set, after applying the MixColumnsSerial (L) operation (a full round from the input set), can be divided into 2^{32} sub-set so that each has one active byte and the other are constant. Consequently, after another 3 rounds, each bit of the output will have zero-sum as mentioned previously in the 3-round distinguisher section.

$$\begin{bmatrix} \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{A} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \bar{\mathcal{A}} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \bar{\mathcal{A}} \end{bmatrix} \xRightarrow{SR\circ S\circ X} \begin{bmatrix} \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \bar{\mathcal{A}} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \bar{\mathcal{A}} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \end{bmatrix} \xRightarrow{L} 2^{32}\times \left\{ \begin{bmatrix} \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \end{bmatrix} \xRightarrow{3R} \begin{bmatrix} \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \end{bmatrix} \right\}$$

MILP for the New 4-round Distinguisher. Our model can be started at the MixColumnsSerial (L) operation of the first round, therefore, we can use the accurate model for the propagation of the BDP described in Section 8.4.1. The first column of the state matrix (in the form of $\boldsymbol{x}$) will be multiplied by M_P. Since the last two element of the vector $\boldsymbol{x}$ are equal, we can express the multiplication operation $\boldsymbol{y} = M_P \times \boldsymbol{x}$ as $\boldsymbol{y} = \hat{M}_P \times \hat{\boldsymbol{x}}$ where $\hat{\boldsymbol{x}} = (x_0, x_1, x_2, x_3, x_4)^T$ and $\hat{M}_P$ can be derived as follows.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{02} & \texttt{01} & \texttt{04} \\ \texttt{08} & \texttt{0e} & \texttt{07} & \texttt{09} & \texttt{06} & \texttt{11} \\ \texttt{22} & \texttt{3b} & \texttt{1f} & \texttt{25} & \texttt{18} & \texttt{42} \\ \texttt{84} & \texttt{e4} & \texttt{79} & \texttt{9b} & \texttt{67} & \texttt{0b} \\ \texttt{16} & \texttt{99} & \texttt{ef} & \texttt{6f} & \texttt{90} & \texttt{4b} \\ \texttt{96} & \texttt{cb} & \texttt{d2} & \texttt{79} & \texttt{24} & \texttt{a7} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_4 \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{02} & \texttt{01} \oplus \texttt{04} \\ \texttt{08} & \texttt{0e} & \texttt{07} & \texttt{09} & \texttt{06} \oplus \texttt{11} \\ \texttt{22} & \texttt{3b} & \texttt{1f} & \texttt{25} & \texttt{18} \oplus \texttt{42} \\ \texttt{84} & \texttt{e4} & \texttt{79} & \texttt{9b} & \texttt{67} \oplus \texttt{0b} \\ \texttt{16} & \texttt{99} & \texttt{ef} & \texttt{6f} & \texttt{90} \oplus \texttt{4b} \\ \texttt{96} & \texttt{cb} & \texttt{d2} & \texttt{79} & \texttt{24} \oplus \texttt{a7} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

$$= \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{02} & \texttt{05} \\ \texttt{08} & \texttt{0e} & \texttt{07} & \texttt{09} & \texttt{17} \\ \texttt{22} & \texttt{3b} & \texttt{1f} & \texttt{25} & \texttt{5a} \\ \texttt{84} & \texttt{e4} & \texttt{79} & \texttt{9b} & \texttt{6c} \\ \texttt{16} & \texttt{99} & \texttt{ef} & \texttt{6f} & \texttt{db} \\ \texttt{96} & \texttt{cb} & \texttt{d2} & \texttt{79} & \texttt{83} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \triangleq \hat{M}_P \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

Consequently, we use the primitive matrix of $\hat{M}_P$ for the first column and the primitive matrix of M_P for other columns. Regarding the intermediate linear layers, we use the disjointed representation with discarding the invalid trails approach presented at Section 8.4.2. The result of solving the model is that a valid division trail that ends at a unit vector does not exist for any output bits, *i.e.*, each output bit after 4 rounds will have zero-sum. It should be mentioned that the model of the first linear layer using the disjointed representation without discarding the invalid trails leads some bits to be imbalanced.

5-round Integral Distinguisher. Similar to the new 4-round one, we employed the same property of the matrix M_P to build the 5-round distinguisher. We firstly construct a set of 2^{280} chosen plaintexts where the last two elements of the diagonal are active and equal (denoted as $\bar{\mathcal{A}}$), and the other elements of the state matrix are active. This set can be divided, after the first round, into 2^{232} sub-sets such that every sub-set has 6 bytes active at specific positions as shown below. Therefore, each sub-set can be considered as an input to 4-round distinguisher that exploit the structure of the round function.

$$\begin{bmatrix} \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \bar{\mathcal{A}} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \bar{\mathcal{A}} \end{bmatrix} \xRightarrow{SR\circ S\circ X} \begin{bmatrix} \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \bar{\mathcal{A}} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \\ \bar{\mathcal{A}} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} & \mathcal{A} \end{bmatrix} \xRightarrow{L} 2^{232} \times \left\{ \begin{bmatrix} \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{A} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{A} \\ \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{A} & \mathcal{C} & \mathcal{C} & \mathcal{C} \\ \mathcal{C} & \mathcal{C} & \mathcal{C} & \mathcal{A} & \mathcal{C} & \mathcal{C} \end{bmatrix} \xRightarrow{4R} \begin{bmatrix} \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} & \mathcal{B} \end{bmatrix} \right\}$$

MILP for the 5-round distinguisher. We have followed the same steps as modelling the 4-round distinguisher to model the 5-round one, where we use the primitive matrix of $\hat{M}_P$ for the first column multiplication in the first round at which the model starts and the primitive matrix of M_P for the other columns. The result of solving the model indicates that each output bit after 5 rounds is balanced.

Other Experiments. We have employed our search approach (Section 8.4.2) to build a regular 5-round distinguisher that does not exploit the previous property of the matrix. We verified that this kind of distinguisher does not exist even when the number of active bits are 287 bits. Also, we have tried to reduce the number of active bits in both the regular and the new 4-round distinguisher by setting one of the active bits to constant and resolving the model. We verified that a distinguisher using less number of active bits does not exist.

8.6 Summary

In this chapter, we proposed a new MILP model for the propagation of the BDP through large non-bit-permutation linear layers. With the help of our model, we improved the previous 3- and 4-round integral distinguishers of Kuznyechik block cipher and the 4-round one of PHOTON's internal permutation (P_{288}). We also found, for the first time, two 4-round integral distinguishers for Kalyna block cipher and a 5-round integral distinguisher for PHOTON's internal permutation (P_{288}).

Chapter 9

Summary and Future Research Directions

9.1 Summary of Contributions

In this section, we give a brief summary of the contributions accomplished in this thesis. We have assessed the security of several block ciphers including CRAFT, SPARX-128/256, Bel-T, T-TWINE, Kuznyechik, and Kalyna. Moreover, we have proposed some modelling techniques that help to automate the search process for finding distinguishers in differential and integral cryptanalysis. In the following, we report the major contributions of this thesis.

In Chapter 3, we studied the security of the lightweight tweakable block cipher CRAFT against related-key differential cryptanalysis. Firstly, we utilized the simple key schedule of CRAFT to build 17 repeatable 2-round related-key differential characteristics that hold with the probability of 2^{-2}. Next, we mounted a key recovery attack on full-round CRAFT using 2^{31} queries to the encryption oracle and 2^{85} encryptions, and 2^{41} 64-bit blocks of memory. Moreover, we sped up the key recovery attack by using 8 different related-key differential characteristics (with 8 related-key differences) in order to recover

96 bits from the secret master key. Then, we got the whole master key by testing the right 96-bit key along with the remaining 32 bits of the key using a plaintext/ciphertext pair. Finally, we omitted the exhaustive search step from the previous attack and recovered the whole master key with $2^{36.09}$ queries to the encryption oracle and only 11 full-round encryptions (instead of 2^{32} in the above attack) using 16 different related-key differential characteristics (with 16 related-key differences).

In Chapter 4, we analyzed the block cipher SPARX-128/256 against impossible differential cryptanalysis. Firstly, we presented two 20-round impossible differential distinguishers. Then, we used them to launch a key recovery attack against 24 rounds. During this attack, we considered the cascaded S-boxes in the same branch as a one large S-box, hence, we constructed 3 look-up tables corresponding to the 3 branches involved in the analysis phase to speed up the attack. Also, we employed the two distinguishers concurrently to reduce the data complexity by a factor of two and enhance the time complexity.

In Chapter 5, we addressed the limitation of the current MILP model for the propagation of the XOR difference through the modular operations, *i.e.,* modular addition and modular subtraction. The current model assumes that the inputs to the modular operations are independent. However, we showed by examples that this assumption did not hold. Accordingly, we proposed a new MILP model for the XOR difference through the modular operations taking into account the dependency between the convective operations. We then utilized the new model to assess the national standard of the Republic of Belarus — Bel-T-256, against differential cryptanalysis. In particular, we reported a 3-round distinguisher with the probability of 2^{-111}. Since the secret key in the Bel-T round function is mixed using modular additions, Bel-T is not a key-alternating cipher and the probability of the distinguisher may drop to zero due to the used key. Therefore, we presented a procedure to find the secret keys which we can use the distinguisher with. Based on our analysis, this distinguisher is valid for $2^{252.2}$ (out of 2^{256}) secret keys. We then employed this distinguisher to attack $4\frac{1}{7}$-round Bel-T-256.

In Chapter 6, we continued investigating the security of Bel-T-256, but this time against integral cryptanalysis using the bit-based division property (BDP). Firstly, we proposed MILP models for the propagation of the BDP through modular additions with a constant and modular subtractions. We also conducted some experiments on a toy cipher to validate our models. Then, we utilized these models, in addition to the MILP models for the modular addition and the Bel-T's S-box, to search for the longest integral distinguisher. As a result, we found several 2-round distinguishers. Finally, we employed two 2-round distinguishers to perform key recovery attacks against $3\frac{2}{7}$-round Bel-T-256 and $3\frac{6}{7}$-Round Bel-T-256.

In Chapter 7, we studied the security of the tweakable block cipher T-TWINE against integral cryptanalysis. Firstly, we searched for the longest integral distinguisher using the BDP in chosen tweak, chosen tweak-plaintext, and chosen tweak-ciphertext attack settings. As a result, we found two 11-round integral distinguishers using a tweak with only one active nibble in the chosen tweak setting. Also, we found several 19-round integral distinguishers in both chosen tweak-plaintext and chosen tweak-ciphertext settings. This allowed us to attack an extra three rounds more than TWINE block cipher.Precisely, We employed meet-in-the-middle and partial-sum techniques to convert the best distinguishing attack to key recovery attacks against 26 (27) out of 36 rounds of T-TWINE-80 (T-TWINE-128) by appending 7 (8) rounds after the disntinguisher. Finally, we extended the attack one more round without using the full codebook of the plaintext by prepending one round before the distinguisher and using dynamically chosen plaintexts. Therefore, we were able to attack 27 and 28 rounds of T-TWINE-80 and T-TWINE-128.

In Chapter 8, we proposed a new MILP model for the propagation of the BDP through large non-bit-permutation linear layers. The previous models are either inaccurate or inefficient for large linear layers. The new model employs the one-to-one map between the valid division trails through the primitive matrix represented the linear layer and its invertible sub-matrices. With the help of our model, we improved the previous 3- and 4-round integral distinguishers of Kuznyechik block cipher and the 4-round one of PHOTON's internal permutation (P_{288}). We also found, for the first time, two 4-round integral distinguishers for Kalyna block cipher and a 5-round integral distinguisher for PHOTON's internal permutation (P_{288}).

9.2 Future Work

In this section, we discuss some topics that would be of interest for future research.

- We have utilized MILP models in differential and integral cryptanalysis to automate finding distinguishers. It would be of interest to apply the same concept to other cryptanalysis techniques that are not automated yet such as differential-linear cryptanalysis. Furthermore, finding the best differential is more important than finding the best differential trail in the context of differential cryptanalysis as mentioned in Chapter 2. However, the current automation tools target the best differential trail only. Finding the best differential is still an open problem. It would be of interest to develop an automation tool that helps to find the best differential. Moreover, the current automation tools focus on propagating the XOR differences ($\Delta X_{input} = X_{input} \oplus X'_{input}$). However, there are other forms of the differences may be more powerful such as the modular differences ($\Delta X_{input} = X_{input} \boxminus X'_{input}$) in the context of ARX-based block ciphers.
- Recently, Gohr highlighted in [46] a new research direction in theoretical cryptanalysis. He showed that machine learning (ML) could potentially be very useful to finding cryptographic distinguishers. These distinguishers could use features that are invisible to purely traditional distinguishers. Firstly, he trained several neural network classifiers to distinguish the output of round-reduced SPECK block cipher with a given input difference from random data. Then, he used one of these clas-

sifiers to mount a key recovery attack. It would be interesting to apply the new ML-based cryptanalysis to other symmetric-key primitives. Also, the presented classifiers exploit differential properties of SPECK. Therefore, it would be interesting to investigate the possibility to exploit other properties of block ciphers such as the integral properties.

- Format-Preserving Encryption (FPE) [8] is a kind of encryption algorithm in which the plaintext and the ciphertext have the same format, *e.g.,* encrypting a 16-digit credit card number so that the ciphertext is another 16-digit credit card number. FFX [9], VAES3 [101], and BPS [20] are examples for FPE schemes. Generally speaking, FPE schemes and block ciphers rely on the same design strategies. Therefore, it would be interesting to investigate the ability to apply the attacks presented in this thesis to FPE schemes.

Bibliography

[1] A. Abdelkhalek, Y. Sasaki, Y. Todo, M. Tolba, and A. Youssef. MILP modeling for (large) s-boxes to optimize probability of differential characteristics. *IACR Transactions on Symmetric Cryptology*, 2017(4):99–129, Dec. 2017.

[2] A. Abdelkhalek, M. Tolba, and A. M. Youssef. Related-key Differential Attack on Round-Reduced Bel-T-256. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 101(5):859–862, 2018.

[3] R. AlTawy, M. ElSheikh, A. M. Youssef, and G. Gong. Lelantos: A Blockchain-Based Anonymous Physical Delivery System. In *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pages 15–1509. IEEE, 2017.

[4] S. Banik, S. K. Pandey, T. Peyrin, Y. Sasaki, S. M. Sim, and Y. Todo. GIFT: A Small Present. In W. Fischer and N. Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 321–345. Springer, 2017.

[5] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers. The SIMON and SPECK lightweight block ciphers. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[6] 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 M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[7] C. Beierle, G. Leander, A. Moradi, and S. Rasoolzadeh. CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks. *IACR Transactions on Symmetric Cryptology*, 2019(1):5–45, Mar. 2019.

[8] M. Bellare, T. Ristenpart, P. Rogaway, and T. Stegers. Format-Preserving Encryption. In M. J. Jacobson, V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography — SAC 2009*, volume 5867 of *Lecture Notes in Computer Science*, pages 295–312. Springer, 2009.

[9] M. Bellare, P. Rogaway, and T. Spies. The FFX mode of operation for format-preserving encryption. *NIST submission*, 2010.

[10] Data Encryption and Integrity Algorithms. Preliminary State Standard of Republic of Belarus (STBP 34.101.312011), 2011. http://apmi.bsu.by/assets/files/std/belt-spec27.pdf.

[11] E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In J. Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[12] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology – CRYPT0 '90*, volume 537 of *Lecture Notes in Computer Science*, pages 2–21. Springer, 1991.

[13] E. Biham and A. Shamir. *Differential Cryptanalysis of the Data Encryption Standard*. Springer, 1993.

[14] A. Biryukov, D. Khovratovich, and L. Perrin. Multiset-Algebraic Cryptanalysis of Reduced Kuznyechik, Khazad, and secret SPNs. *IACR Transactions on Symmetric Cryptology*, 2016(2):226–247, Feb. 2017.

[15] A. Biryukov and A. Shamir. Structural Cryptanalysis of SASAS. In B. Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 395–405. Springer, 2001.

[16] A. Biryukov and V. Velichkov. Automatic Search for Differential Trails in ARX Ciphers. In J. Benaloh, editor, *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *Lecture Notes in Computer Science*, pages 227–250. Springer, 2014.

[17] A. Bogdanov. *Analysis and Design of Block Cipher Constructions.* PhD thesis, Ruhr University Bochum, 2009.

[18] 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 P. Paillier and I. Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[19] A. Bogdanov and V. Rijmen. Linear hulls with correlation zero and linear cryptanalysis of block ciphers. *Designs, Codes and Cryptography*, 70(3):369–383, Mar 01, 2014.

[20] E. Brier, T. Peyrin, and J. Stern. BPS: a format-preserving encryption proposal. *NIST submission*, 2010.

[21] Z. Chu, H. Chen, X. Wang, L. Li, X. Dong, Y. Ding, and Y. Hao. Improved integral attacks without full codebook. *IET Information Security*, 12(6):513–520, 2018.

[22] T. Cui, S. Chen, K. Jia, K. Fu, and M. Wang. New Automatic Search Tool for Impossible Differentials and Zero-Correlation Linear Approximations. Cryptology ePrint Archive, Report 2016/689, 2016. https://eprint.iacr.org/2016/689.

[23] J. Daemen, L. Knudsen, and V. Rijmen. The block cipher Square. In E. Biham, editor, *Fast Software Encryption – FSE 1997*, volume 1267 of *Lecture Notes in Computer Science*, pages 149–165. Springer, 1997.

[24] J. Daemen and V. Rijmen. Probability distributions of Correlation and Differentials in Block Ciphers. *Journal of Mathematical Cryptology JMC*, 1(3):221–242, 2007.

[25] Y. De Mulder. *White-Box Cryptography: Analysis of White-Box AES Implementations (White-Box Cryptografie: Analyse van White-Box AES implementaties)*. PhD thesis, KU Leuven, 2014.

[26] FIPS-46: Data Encryption Standard (DES). National Institute of Standards and Technology, 1979. https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf.

[27] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[28] D. Dinu, L. Perrin, A. Udovenko, V. Velichkov, J. Großschädl, and A. Biryukov. Design Strategies for ARX with Provable Bounds: SPARX and LAX. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 484–513. Springer, 2016.

[29] V. Dolmatov. GOST R 34.12-2015: Block Cipher ”Kuznyechik”. RFC 7801, RFC Editor, 3 2016. https://tools.ietf.org/html/rfc7801.

[30] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray. Advanced Encryption Standard (AES). Federal Inf. Process. Stds. (NIST FIPS),

National Institute of Standards and Technology, Gaithersburg, MD, 2001. https://doi.org/10.6028/NIST.FIPS.197.

[31] M. ElSheikh, A. Abdelkhalek, and A. M. Youssef. On MILP-Based Automatic Search for Differential Trails Through Modular Additions with Application to Bel-T. In J. Buchmann, A. Nitaj, and T. Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2019*, volume 11627 of *Lecture Notes in Computer Science*, pages 273–296. Springer, 2019.

[32] M. ElSheikh, J. Clark, and A. M. Youssef. Short Paper: Deploying PayWord on Ethereum. In A. Bracciali, J. Clark, F. Pintore, P. B. Rønne, and M. Sala, editors, *Financial Cryptography and Data Security – FC 2019*, volume 11599 of *Lecture Notes in Computer Science*, pages 82–90. Springer, 2020.

[33] M. ElSheikh, M. Tolba, and A. M. Youssef. Impossible Differential Attack on Reduced Round SPARX-128/256. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E101.A(4):731–733, 2018.

[34] M. ElSheikh, M. Tolba, and A. M. Youssef. Integral Attacks on Round-Reduced Bel-T-256. In C. Cid and M. J. Jacobson Jr., editors, *Selected Areas in Cryptography – SAC 2018*, volume 11349 of *Lecture Notes in Computer Science*, pages 73–91. Springer, 2019.

[35] M. ElSheikh and A. M. Youssef. Related-Key Differential Cryptanalysis of Full Round CRAFT. In S. Bhasin, A. Mendelson, and M. Nandi, editors, *Security, Privacy, and Applied Cryptography Engineering – SPACE 2019*, volume 11947 of *Lecture Notes in Computer Science*, pages 50–66. Springer, 2019.

[36] M. ElSheikh and A. M. Youssef. A cautionary note on the use of Gurobi for cryptanalysis. Cryptology ePrint Archive, Report 2020/1112, 2020. https://eprint.iacr.org/2020/1112.

[37] M. ElSheikh and A. M. Youssef. Integral Cryptanalysis of Reduced-Round Tweakable TWINE. In S. Krenn, H. Shulman, and S. Vaudenay, editors, *Cryptology and Network Security – CANS 2020*, volume 12579 of *Lecture Notes in Computer Science*, pages 485–504. Springer, 2020.

[38] M. ElSheikh and A. M. Youssef. On MILP-based Automatic Search for Bit-Based Division Property for Ciphers with (large) Linear Layers. In J. Baek and S. Ruj, editors, *Information Security and Privacy – ACISP 2021 – 26th Australasian Conference*, Lecture Notes in Computer Science. Springer, 2021. To appear.

[39] H. Feistel, W. A. Notz, and J. L. Smith. Some cryptographic techniques for machine-to-machine data communications. *Proceedings of the IEEE*, 63(11):1545–1554, 1975.

[40] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved Cryptanalysis of Rijndael. In G. Goos, J. Hartmanis, J. van Leeuwen, and B. Schneier, editors, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2001.

[41] K. Fu, M. Wang, Y. Guo, S. Sun, and L. Hu. MILP-Based Automatic Search Algorithms for Differential and Linear Trails for Speck. In T. Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 268–288. Springer, 2016.

[42] H. S. Galal, M. ElSheikh, and A. M. Youssef. An Efficient Micropayment Channel on Ethereum. In C. Pérez-Solà, G. Navarro-Arribas, A. Biryukov, and J. Garcia-Alfaro, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology – CBT 2019*, volume 11737 of *Lecture Notes in Computer Science*, pages 211–218. Springer, 2019.

[43] D. Gérault and P. Lafourcade. Related-Key Cryptanalysis of Midori. In O. Dunkelman and S. K. Sanadhya, editors, *Progress in Cryptology – INDOCRYPT 2016*, volume 10095 of *Lecture Notes in Computer Science*, pages 287–304. Springer, 2016.

[44] D. Gerault, M. Minier, and C. Solnon. Constraint Programming Models for Chosen Key Differential Cryptanalysis. In M. Rueher, editor, *Principles and Practice of Constraint Programming – CP 2016*, volume 9892 of *Lecture Notes in Computer Science*, pages 584–601. Springer, 2016.

[45] H. Gilbert and M. Minier. A Collision Attack on 7 Rounds of Rijndael. In *Proceedings of Third Advanced Encryption Standard Conference*, pages 230–241. National Institute of Standards and Technology, 2000.

[46] A. Gohr. Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning. In A. Boldyreva and D. Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, volume 11693 of *Lecture Notes in Computer Science*, pages 150–179. Springer, 2019.

[47] J. Guo, T. Peyrin, and A. Poschmann. The PHOTON Family of Lightweight Hash Functions. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.

[48] L. Gurobi Optimization. Gurobi Optimizer Reference Manual, 2020. http://www.gurobi.com.

[49] M. Hermelin, J. Y. Cho, and K. Nyberg. Multidimensional Extension of Matsui's Algorithm 2. In O. Dunkelman, editor, *Fast Software Encryption – FSE 2009*, volume 5665 of *Lecture Notes in Computer Science*, pages 209–227. Springer, 2009.

[50] K. Hu, Q. Wang, and M. Wang. Finding Bit-Based Division Property for Ciphers with Complex Linear Layers. *IACR Transactions on Symmetric Cryptology*, 2020 (1):396–424, May 2020.

[51] IBM. IBM ILOG CPLEX 12.10 User's Manual, 2020. https://www.ibm.com/support/knowledgecenter/SSSA5P_12.10.0/COS_KC_home.html.

[52] J. Jean, I. Nikolić, T. Peyrin, and Y. Seurin. Deoxys v1.41. Submitted to CAESAR Competition, 2016. https://competitions.cr.yp.to/round3/deoxysv141.pdf.

[53] P. Jovanovic and I. Polian. Fault-based attacks on the Bel-T block cipher family. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 601–604. IEEE, 2015.

[54] A. A. Kamal and A. M. Youssef. Applications of SAT solvers to AES key recovery from decayed key schedule images. In *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*, pages 216–220. IEEE, 2010.

[55] L. Knudsen. Truncated and higher order differentials. In B. Preneel, editor, *Fast Software Encryption – FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1995.

[56] L. Knudsen. DEAL-a 128-bit block cipher. *complexity*, 258(2):216, 1998.

[57] L. Knudsen and D. Wagner. Integral Cryptanalysis. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption – FSE 2002.*, volume 2365 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2002.

[58] L. R. Knudsen. Block Ciphers. In H. C. A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 152–157. Springer, 2011.

[59] L. R. Knudsen and J. E. Mathiassen. A Chosen-Plaintext Linear Attack on DES. In G. Goos, J. Hartmanis, J. van Leeuwen, and B. Schneier, editors, *Fast Software Encryption – FSE 2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 262–272. Springer, 2001.

[60] T. Krovetz and P. Rogaway. The Software Performance of Authenticated-Encryption Modes. In A. Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.

[61] X. Lai. Higher Order Derivatives and Differential Cryptanalysis. In R. E. Blahut, D. J. Costello, U. Maurer, and T. Mittelholzer, editors, *Communications and Cryptography: Two Sides of One Tapestry*, volume 276 of *The Springer International Series in Engineering and Computer Science (Communications and Information Theory)*, pages 227–233. Springer, 1994.

[62] X. Lai and J. L. Massey. A Proposal for a New Block Encryption Standard. In I. B. Damgård, editor, *Advances in Cryptology — EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 389–404. Springer, 1991.

[63] X. Lai, J. L. Massey, and S. Murphy. Markov Ciphers and Differential Cryptanalysis. In D. W. Davies, editor, *Advances in Cryptology – EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 17–38. Springer, 1991.

[64] G. Leurent. Analysis of Differential Attacks in ARX Constructions. In X. Wang and K. Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 226–243. Springer, 2012.

[65] H. Lipmaa and S. Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In M. Matsui, editor, *Fast Software Encryption – FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2002.

[66] M. Liskov, R. L. Rivest, and D. Wagner. Tweakable block ciphers. *Journal of cryptology*, 24(3):588–613, 2011.

[67] J. Lu, J. Kim, N. Keller, and O. Dunkelman. Improving the Efficiency of Impossible Differential Cryptanalysis of Reduced Camellia and MISTY1. In T. Malkin, editor,

Topics in Cryptology – CT-RSA 2008, volume 4964 of *Lecture Notes in Computer Science*, pages 370–386. Springer, 2008.

[68] S. Lucks. The Saturation Attack — A Bait for Twofish. In M. Matsui, editor, *Fast Software Encryption – FSE 2001*, volume 2355 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[69] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In T. Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 386–397. Springer, 1994.

[70] M. Matsui and A. Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In R. A. Rueppel, editor, *Advances in Cryptology – EUROCRYPT' 92*, volume 658 of *Lecture Notes in Computer Science*, pages 81–91. Springer, 1993.

[71] E. J. McCluskey Jr. Minimization of boolean functions. *Bell system technical Journal*, 35(6):1417–1444, 1956.

[72] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1st edition, 1996.

[73] N. Mouha, Q. Wang, D. Gu, and B. Preneel. Differential and linear cryptanalysis using Mixed-Integer Linear Programming. In C.-K. Wu, M. Yung, and D. Lin, editors, *Information Security and Cryptology – Inscrypt 2011*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[74] R. Oliynykov, I. Gorbenko, O. Kazymyrov, V. Ruzhentsev, O. Kuznetsov, Y. Gorbenko, O. Dyrda, V. Dolgov, A. Pushkaryov, R. Mordvinov, and D. Kaidalov. A New Encryption Standard of Ukraine: The Kalyna Block Cipher. Cryptology ePrint Archive, Report 2015/650, 2015. https://eprint.iacr.org/2015/650.

[75] T. Peyrin and Y. Seurin. Counter-in-Tweak: Authenticated Encryption Modes for Tweakable Block Ciphers. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016*, volume 9814 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2016.

[76] B. Preneel. Hash Functions. In H. C. A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 543–553. Springer, 2011.

[77] B. Preneel. MAC Algorithms. In H. C. A. van Tilborg and S. Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 742–748. Springer, 2011.

[78] W. V. O. Quine. A way to simplify truth functions. *The American Mathematical Monthly*, 62(9):627–631, 1955.

[79] R. L. Rivest, A. Shamir, and L. L. Adleman. A Method for Obtaining Public Key Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21, 1978.

[80] P. Rogaway. Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC. In P. J. Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[81] K. Sakamoto, K. Minematsu, N. Shibata, M. Shigeri, H. Kubo, Y. Funabiki, A. Bogdanov, S. Morioka, and T. Isobe. Tweakable TWINE: Building a Tweakable Block Cipher on Generalized Feistel Structure. In N. Attrapadung and T. Yagi, editors, *Advances in Information and Computer Security – IWSEC 2019*, volume 11689 of *Lecture Notes in Computer Science*, pages 129–145. Springer, 2019.

[82] Y. Sasaki and Y. Todo. New Impossible Differential Search Tool from Design and Cryptanalysis Aspects. In J.-S. Coron and J. B. Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, volume 10212 of *Lecture Notes in Computer Science*, pages 185–215. Springer, 2017.

[83] Y. Sasaki and L. Wang. Meet-in-the-Middle Technique for Integral Attacks against Feistel Ciphers. In L. R. Knudsen and H. Wu, editors, *Selected Areas in Cryptography – SAC 2012*, volume 10624 of *Lecture Notes in Computer Science*, pages 234–251. Springer, 2013.

[84] A. A. Selçuk. On Probability of Success in Linear and Differential Cryptanalysis. *Journal of Cryptology*, 21(1):131–147, Jan 01, 2008.

[85] C. E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[86] V. Shishkin, D. Dygin, I. Lavrikov, G. Marshalko, V. Rudskoy, and D. Trifonov. Low-weight and hi-end: Draft Russian encryption standard. In *3rd Workshop on Current Trends in Cryptology - CTCrypt 2014*, pages 183–188, 2014.

[87] B. Sun, M. Liu, J. Guo, L. Qu, and V. Rijmen. New Insights on AES-Like SPN Ciphers. In M. Robshaw and J. Katz, editors, *Advances in Cryptology – CRYPTO 2016*, volume 9814 of *Lecture Notes in Computer Science*, pages 605–624. Springer, 2016.

[88] L. Sun, W. Wang, R. Liu, and M. Wang. MILP-Aided Bit-Based Division Property for ARX-Based Block Cipher. Cryptology ePrint Archive, Report 2016/1101, 2016. https://eprint.iacr.org/2016/1101.

[89] L. Sun, W. Wang, and M. Wang. Automatic Search of Bit-Based Division Property for ARX Ciphers and Word-Based Division Property. In T. Takagi and T. Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 128–157. Springer, 2017.

[90] L. Sun, W. Wang, and M. Q. Wang. MILP-aided bit-based division property for primitives with non-bit-permutation linear layers. *IET Information Security*, 14: 12–20(8), January 2020.

[91] S. Sun, L. Hu, M. Wang, P. Wang, K. Qiao, X. Ma, D. Shi, L. Song, and K. Fu. Towards Finding the Best Characteristics of Some Bit-oriented Block Ciphers and Automatic Enumeration of (Related-key) Differential and Linear Characteristics with Predefined Properties. Cryptology ePrint Archive, Report 2014/747, 2014. https://eprint.iacr.org/2014/747.

[92] S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song. Automatic Security Evaluation and (Related-key) Differential Characteristic Search: Application to SIMON, PRESENT, LBlock, DES(L) and Other Bit-Oriented Block Ciphers. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014.

[93] T. Suzaki and K. Minematsu. Improving the Generalized Feistel. In S. Hong and T. Iwata, editors, *Fast Software Encryption – FSE 2010*, volume 6147 of *Lecture Notes in Computer Science*, pages 19–39. Springer, 2010.

[94] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi. TWINE: A Lightweight Block Cipher for Multiple Platforms. In L. R. Knudsen and H. Wu, editors, *Selected Areas in Cryptography – SAC 2012*, volume 7707 of *Lecture Notes in Computer Science*, pages 339–354. Springer, 2013.

[95] Y. Todo. Structural Evaluation by Generalized Integral Property. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, 2015.

[96] Y. Todo. Integral Cryptanalysis on Full MISTY1. *Journal of Cryptology*, 30(3): 920–959, 2017.

[97] Y. Todo, T. Isobe, Y. Hao, and W. Meier. Cube Attacks on Non-Blackbox Polynomials Based on Division Property. *IEEE Transactions on Computers*, 67(12): 1720–1736, 2018.

[98] Y. Todo and M. Morii. Bit-Based Division Property and Application to Simon Family. In T. Peyrin, editor, *Fast Software Encryption – FSE 2016*, volume 9783 of *Lecture Notes in Computer Science*, pages 357–377. Springer, 2016.

[99] M. Tolba, A. Abdelkhalek, and A. M. Youssef. Multidimensional Zero-Correlation Linear Cryptanalysis of Reduced Round SPARX-128. In C. Adams and J. Camenisch, editors, *Selected Areas in Cryptography – SAC 2017*, volume 10719 of *Lecture Notes in Computer Science*, pages 423–441. Springer, 2018.

[100] M. Tolba, M. ElSheikh, and A. M. Youssef. Impossible Differential Cryptanalysis of Reduced-Round Tweakable TWINE. In A. Nitaj and A. Youssef, editors, *Progress in Cryptology – AFRICACRYPT 2020*, volume 12174 of *Lecture Notes in Computer Science*, pages 91–113. Springer, 2020.

[101] J. Vance. VAES3 scheme for FFX: An addendum to The FFX mode of operation for Format Preserving Encryption. *NIST submission*, 2011.

[102] D. Wagner. The Boomerang Attack. In L. Knudsen, editor, *Fast Software Encryption – FSE 1999*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

[103] G. Wang, N. Keller, and O. Dunkelman. The Delicate Issues of Addition with Respect to XOR Differences. In C. Adams, A. Miri, and M. Wiener, editors, *Selected Areas in Cryptography – SAC 2007*, volume 4876 of *Lecture Notes in Computer Science*, pages 212–231. Springer, 2007.

[104] 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 J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 648–678. Springer, 2016.

[105] H. Zhang and W. Wu. Structural Evaluation for Generalized Feistel Structures and Applications to LBlock and TWINE. In A. Biryukov and V. Goyal, editors, *Progress in Cryptology – INDOCRYPT 2015*, volume 9462 of *Lecture Notes in Computer Science*, pages 218–237. Springer, 2015.

[106] W. Zhang and V. Rijmen. Division cryptanalysis of block ciphers with a binary diffusion layer. *IET Information Security*, 13:87–95(8), March 2019.

Appendix A

Recovery of 80-bit Keys of T-TWINE-80 Attack

During the key recovery attack against T-TWINE-80 in Chapter 7, we have got 2^{72} 76-bit candidates of the 19 round keys $RK^{26}_{[0,1,2,3,4,5,6,7]}, RK^{25}_{[0,1,2,3,4,5,7]}, RK^{24}_{[6,7]}, RK^{23}_{0}, RK^{22}_{5}$ as shown in Section 7.4.1. In this appendix, we describe how we can transform them to the 80-bit candidates of the master key.

Based on the key schedule of T-TWINE-80, these 19 round keys can be expressed as:

$$RK^{23}_{0} = V_1 \oplus CL_9 \oplus CH_{12} \quad \text{(A.1)}$$

$$RK^{25}_{0} = V_2 \oplus CL_{11} \oplus CH_{14} \quad \text{(A.2)}$$

$$RK^{26}_{0} = V_3 \oplus CL_{12} \oplus CH_{15} \quad \text{(A.3)}$$

$$RK^{25}_{4} = V_4 \oplus CL_{14} \oplus CH_{17} \quad \text{(A.4)}$$

$$RK^{26}_{4} = V_5 \oplus CL_{15} \oplus CH_{18} \quad \text{(A.5)}$$

$$RK^{22}_{5} = V_6 \oplus CL_{16} \oplus CH_{19} \quad \text{(A.6)}$$

$$RK^{26}_{3} = V_7 \oplus CL_{18} \oplus CH_{21} \quad \text{(A.7)}$$

$$RK^{25}_{3} = V_8 \oplus CL_{17} \oplus CH_{20} \quad \text{(A.8)}$$

$$RK_5^{26} = V_9 \oplus CL_{20} \oplus CH_{23} \tag{A.9}$$

$$RK_1^{26} = V_{10} \oplus CL_{22} \oplus CH_{25} \tag{A.10}$$

$$RK_1^{25} = V_{11} \oplus CL_{21} \oplus CH_{24} \tag{A.11}$$

$$RK_2^{26} = V_{12} \tag{A.12}$$

$$RK_2^{25} = V_{13} \tag{A.13}$$

$$RK_5^{25} = V_{14} \oplus CL_{19} \oplus CH_{22} \tag{A.14}$$

$$RK_7^{24} = V_{15} \tag{A.15}$$

$$RK_7^{26} = V_2 \oplus CL_{11} \oplus CH_{14} \oplus S(V_{16} \oplus CL_6 \oplus CH_9 \oplus S(V_{11}) \oplus S(V_{15})) \tag{A.16}$$

$$RK_6^{24} = V_{17} \oplus CL_3 \oplus CH_6 \oplus S(V_7) \oplus S(V_{16} \oplus CL_6 \oplus CH_9 \oplus S(V_{11})) \oplus CL_{23} \tag{A.17}$$

$$RK_6^{26} = V_{18} \oplus CL_5 \oplus CH_8 \oplus S(V_9) \oplus S(V_{12}) \oplus CL_{25} \tag{A.18}$$

$$RK_7^{25} = V_{19} \oplus CL_{10} \oplus CH_{13} \oplus S(V_{18} \oplus CL_5 \oplus CH_8 \oplus S(V_9) \oplus S(V_{12})) \tag{A.19}$$

where $CL_i = 0||CON_L^i$ and $CH_i = 0||CON_H^i$ are predefined constants. The variables $V_1, \ldots, V_{19}$ are expressed as follows:

$$V_9 = K_{15} \oplus CH_3 \oplus S(V_5) \oplus S(V_{17} \oplus CL_3 \oplus CH_6 \oplus S(V_7)) \tag{A.20}$$

$$V_8 = K_3 \oplus S(V_3) \oplus S(K_{15} \oplus CH_3 \oplus S(V_5)) \tag{A.21}$$

$$V_4 = K_{10} \oplus S(V_1) \oplus S(K_3 \oplus S(V_3)) \tag{A.22}$$

$$V_2 = K_{17} \oplus S(V_{16})) \oplus S(K_{10} \oplus S(V_1)) \tag{A.23}$$

$$\begin{split} V_{12} = {} & K_5 \oplus S(V_{17}) \oplus S(K_{17} \oplus S(V_{16}))) \oplus CL_8 \oplus CH11 \oplus S(V_{17} \oplus CL_3 \\ & \oplus CH_6 \oplus S(V_7) \oplus S(V_{16}) \oplus CL_6 \oplus CH_9 \oplus S(V_{11}))) \end{split} \tag{A.24}$$

$$V_{18} = K_{12} \oplus S(K_5 \oplus S(V_{17})) \tag{A.25}$$

$$V_{10} = K_0 \oplus CL_2 \oplus CH_5 \oplus S(V_8) \oplus S(V_{18} \oplus CL_5 \oplus CH_8 \oplus S(V_9)) \tag{A.26}$$

$$V_{14} = K_{11} \oplus CH_2 \oplus S(V_4) \oplus S(K_0 \oplus CL_2 \oplus CH_5 \oplus S(V_8)) \tag{A.27}$$

$$V_6 = K_{18} \oplus S(V_2) \oplus S(K_{11} \oplus CH_2 \oplus S(V_4)) \tag{A.28}$$

$$V_{15} = V_1 \oplus CL_9 \oplus CH12 \oplus S(A) \oplus CL_4 \oplus CH_7 \oplus S(V_{14}) \oplus S(V_{13})) \tag{A.29}$$

$$V_{11} = K_{19} \oplus CL_1 \oplus CH_4 \oplus S(V_6) \oplus S(A) \oplus CL_4 \oplus CH_7 \oplus S(V_{14})) \tag{A.30}$$

$$V_7 = B \oplus S(K_{19} \oplus CL_1 \oplus CH_4 \oplus S(V_6)) \tag{A.31}$$

$$V_5 = K_{14} \oplus S(V_{19}) \oplus S(B) \tag{A.32}$$

$$V_{13} = C \oplus CL_7 \oplus CH10 \oplus S(V_{10}) \tag{A.33}$$

$$V_3 = K_2 \oplus S(C) \oplus S(K_{14} \oplus S(V_{19})) \tag{A.34}$$

$$V_1 = K_9 \oplus S(A)) \oplus S(K_2 \oplus S(C)) \tag{A.35}$$

$$V_{16} = K_{16} \oplus S(K_9 \oplus S(A)) \tag{A.36}$$

$$V_{17} = K_4 \oplus S(K16) \tag{A.37}$$

$$V_{19} = K_{13} \oplus S(V_{18}) \oplus S(K_6 \oplus S(K_5 \oplus S(V_{17}) \oplus S(K_{17} \oplus S(V_{16}))))) \tag{A.38}$$

$$\begin{aligned} B = {} & K_7 \oplus CH_1 \oplus S(K_6 \oplus S(K_5 \oplus S(V_{17}) \oplus S(K_{17} \oplus S(V_{16})))) \oplus S(K_{18} \\ & \oplus S(V_2))) \end{aligned} \tag{A.39}$$

$$C = K_1 \oplus S(K_0) \oplus S(K_{13} \oplus S(V_{18})) \tag{A.40}$$

$$A = K_8 \oplus S(K_1 \oplus S(K_0) \tag{A.41}$$

Therefore, we can compute the values of the variables $V_1, \ldots, V_{19}$ directly from equations (A.1)–(A.19). Hence, we substitute their values into the equations (A.20)–(A.41). Thus, it is easy to obtain the values of $K_{15}, K_3, K_{10}, K_{17}, K_5, K_{12}, K_0, K_{11}, K_{18}, A, K_{19}, B, K_{14}, C, K_2, K_9, K_{16},$ K_4 one by one from equations (A.20)–(A.37). Next, we guess the value of K_6 and obtain the values of K_{13}, K_7, K_1, K_8 from equations (A.38)–(A.41).

Appendix B

Gaussian Elimination for the Toy Linear Layer

$$\left[\begin{array}{ccccc|l}
1 & 0 & 0 & 0 & 0 & r_0 \\
1 & 1 & 0 & 0 & 1 & r_1 \\
0 & 1 & 0 & 0 & 0 & r_2 \\
0 & 0 & 1 & 0 & 0 & r_3 \\
0 & 0 & 0 & 1 & 0 & r_4 \\
0 & 1 & 0 & 0 & 1 & r_5 \\
1 & 1 & 0 & 0 & 0 & r_6 \\
0 & 0 & 1 & 1 & 0 & r_7
\end{array}\right] \rightarrow
\left[\begin{array}{ccccc|l}
1 & 0 & 0 & 0 & 0 & r_0 \\
0 & 1 & 0 & 0 & 1 & r_1 + r_0 \\
0 & 1 & 0 & 0 & 0 & r_2 \\
0 & 0 & 1 & 0 & 0 & r_3 \\
0 & 0 & 0 & 1 & 0 & r_4 \\
0 & 1 & 0 & 0 & 1 & r_5 \\
0 & 1 & 0 & 0 & 0 & r_6 + r_0 \\
0 & 0 & 1 & 1 & 0 & r_7
\end{array}\right] \rightarrow
\left[\begin{array}{ccccc|l}
1 & 0 & 0 & 0 & 0 & r_0 \\
0 & 1 & 0 & 0 & 1 & r_1 + r_0 \\
0 & 0 & 0 & 0 & 1 & r_2 + r_1 + r_0 \\
0 & 0 & 1 & 0 & 0 & r_3 \\
0 & 0 & 0 & 1 & 0 & r_4 \\
0 & 0 & 0 & 0 & 0 & r_5 + r_1 + r_0 \\
0 & 0 & 0 & 0 & 1 & r_6 + r_1 \\
0 & 0 & 1 & 1 & 0 & r_7
\end{array}\right] \rightarrow$$

$$\left[\begin{array}{ccccc|l} 1 & 0 & 0 & 0 & 0 & r_0 \\ 0 & 1 & 0 & 0 & 1 & r_1 + r_0 \\ 0 & 0 & 1 & 0 & 0 & r_3 \\ 0 & 0 & 0 & 0 & 1 & r_2 + r_1 + r_0 \\ 0 & 0 & 0 & 1 & 0 & r_4 \\ 0 & 0 & 0 & 0 & 0 & r_5 + r_1 + r_0 \\ 0 & 0 & 0 & 0 & 1 & r_6 + r_1 \\ 0 & 0 & 1 & 1 & 0 & r_7 \end{array}\right] \to \left[\begin{array}{ccccc|l} 1 & 0 & 0 & 0 & 0 & r_0 \\ 0 & 1 & 0 & 0 & 1 & r_1 + r_0 \\ 0 & 0 & 1 & 0 & 0 & r_3 \\ 0 & 0 & 0 & 0 & 1 & r_2 + r_1 + r_0 \\ 0 & 0 & 0 & 1 & 0 & r_4 \\ 0 & 0 & 0 & 0 & 0 & r_5 + r_1 + r_0 \\ 0 & 0 & 0 & 0 & 1 & r_6 + r_1 \\ 0 & 0 & 0 & 1 & 0 & r_7 + r_3 \end{array}\right] \to$$

$$\left[\begin{array}{ccccc|l} 1 & 0 & 0 & 0 & 0 & r_0 \\ 0 & 1 & 0 & 0 & 1 & r_1 + r_0 \\ 0 & 0 & 1 & 0 & 0 & r_3 \\ 0 & 0 & 0 & 1 & 0 & r_4 \\ 0 & 0 & 0 & 0 & 1 & r_2 + r_1 + r_0 \\ 0 & 0 & 0 & 0 & 0 & r_5 + r_1 + r_0 \\ 0 & 0 & 0 & 0 & 1 & r_6 + r_1 \\ 0 & 0 & 0 & 1 & 0 & r_7 + r_3 \end{array}\right] \to \left[\begin{array}{ccccc|l} 1 & 0 & 0 & 0 & 0 & r_0 \\ 0 & 1 & 0 & 0 & 1 & r_1 + r_0 \\ 0 & 0 & 1 & 0 & 0 & r_3 \\ 0 & 0 & 0 & 1 & 0 & r_4 \\ 0 & 0 & 0 & 0 & 1 & r_2 + r_1 + r_0 \\ 0 & 0 & 0 & 0 & 0 & r_5 + r_1 + r_0 \\ 0 & 0 & 0 & 0 & 1 & r_6 + r_1 \\ 0 & 0 & 0 & 0 & 0 & r_7 + r_3 + r_4 \end{array}\right] \to$$

$$\left[\begin{array}{ccccc|l} 1 & 0 & 0 & 0 & 0 & r_0 \\ 0 & 1 & 0 & 0 & 1 & r_1 + r_0 \\ 0 & 0 & 1 & 0 & 0 & r_3 \\ 0 & 0 & 0 & 1 & 0 & r_4 \\ 0 & 0 & 0 & 0 & 1 & r_2 + r_1 + r_0 \\ 0 & 0 & 0 & 0 & 0 & r_5 + r_1 + r_0 \\ 0 & 0 & 0 & 0 & 0 & r_6 + r_2 + r_0 \\ 0 & 0 & 0 & 0 & 0 & r_7 + r_3 + r_4 \end{array}\right] \to \left\{\begin{matrix} r_0 + r_1 + r_5 = \mathbf{0} \\ r_0 + r_2 + r_6 = \mathbf{0} \\ r_3 + r_4 + r_7 = \mathbf{0} \end{matrix}\right\}$$