Security Analysis of Cryptographic Algorithms

Sareh Sadat Emami

Department of Computing, Faculty of Science

Macquarie University

A thesis submitted for the degree of

Doctor of Philosophy

December 2013

Declaration

This thesis is submitted in fulfilment of the requirements of the degree of Doctor of Philosophy at Macquarie University and has not been submitted for a higher degree to any other university or institution. This thesis represents my original work and contributions. I certify that to the best of my knowledge, all sources and assistance received in the preparation of this thesis have been acknowledged.

Sareh Sadat Emami

Contents

Abstract

Design and security analysis of symmetric algorithms are amongst the most important topics in cryptography. This thesis studies cryptanalysis of symmetric algorithms including block ciphers and hash functions. Block ciphers are symmetric-key encryption algorithms employed in many cryptographic systems to provide confidentiality of data. In a secure symmetric encryption algorithm, decryption of the ciphertext should be intractable for parties that do not know the secret key. However, this should be easy for the party who knows the key. Cryptographic hashing algorithms, on the other hand, are predominately used for authentication and integrity verification purposes. Efficient digital signatures are possible when signatures are generated for the message digests instead of for messages themselves. In this case, the security of signatures are tied to the collision resistance of the used hash algorithm.

The thesis provides background material that is necessary to understand the topics covered in the work. In particular, the first two chapters explain the basic design structures and describe analytic tools (also called attacks) that are employed to test the security of cryptographic algorithms. Our contributions, presented in subsequent chapters, are three-fold. First we consider the lightweight block cipher LBlock and analyse its resistance against truncated differential attacks. Next we focus our attention on the Feistel networks and analyse them using the rebound attack. Our approach is illustrated on the Camellia block cipher. The last contribution is an analysis of Grøstel hash function against the preimage attack.

We employ differential probability distributions to improve the truncated differential cryptanalysis and apply key-recovery attacks to the reduced variants of

LBlock. Benefiting from the differential distributions, this technique analyses the security of algorithms with relatively less data compared to other methods.

The truncated differential analysis together with the rebound attack is used to improve attacks on generalised Feistel-SP networks. Then, we study randomness of the reduced-round Camellia block cipher. We also examine hash functions based on Camellia with respect to distinguishing and collision attacks.

Finally, the security of Grøstl hash function is analysed against preimage and multi-target preimage attacks. We exploit the rebound technique to attack the reduced-round Grøstl compression function and find preimages and multi-target preimages for chosen sets of hash values.

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Josef Pieprzyk for his patience, motivation, and immense knowledge. I am also thankful to Dr. Ron Steinfeld and Dr. Cameron McDonald for their invaluable help and insightful comments.

Special thanks to my parents, Hossein and Fereshteh who have supported me spiritually throughout my life.

Finally, I am most grateful to my husband Ali, for his infinite love and support. He has always understood me in difficult times and given me the encouragement to complete this thesis.

ACKNOWLEDGEMENTS

List of Publications

1. Yu Sasaki, Sareh Emami, Deukjo Hong, Ashish Kumar, *Improved Known-Key Distinguishers on Feistel-SP Ciphers and Application to Camellia*. Proceedings of the 17th Australasian Conference on Information Security and Privacy (ACISP' 12), pages 87-100, 9-11 July, 2012, Wollongong, Australia.

2. Sareh Emami, Cameron McDonald, Josef Pieprzyk, and Ron Steinfeld, *Truncated Differential Analysis of Reduced-Round LBlock*. Proceedings of the 12th International Conference on Cryptology and Network Security (CANS' 13), pages 291-308, 20-22 November, 2013, Paraty, Brazil.

In the first paper I played a crucial role in the contribution and running the experiments. While, in paper 2, I played the main role in finding, experimenting and writing.

LIST OF PUBLICATIONS

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

1

Introduction

Information security is the practice of protecting information in the presence of adversaries, and allows for secure communication between two or more parties. Cryptography is the basic tool used to design and analyse security protocols in a communication system. More generally, it is applied to achieve the major security goals such as confidentiality, data integrity, authentication, and non-repudiation.

Cryptography is ubiquitous in our everyday life. Every time we connect to the Internet to check a bank account, send/receive an email, or shop online, the browser uses appropriate cryptographic algorithms on our behalf. It is not just limited to the online activities. Smart-cards, such as credit cards or identification tags, use cryptographic techniques to authenticate the owner's identity. Many cryptographic algorithms are applied in mobile SIM cards, not only to provide authentication but also for message integrity and data confidentiality. Therefore, designing novel cryptographic algorithms is an active research topic. To gain confidence in the security of these algorithms their analysis is always as important as their designs.

Cryptographic Primitives

Low-level cryptographic algorithms which are used to build security systems are known as cryptographic primitives. Cryptographic primitives are classified into three following categories [75]:

1. *Unkeyed algorithms* are keyless primitives, which have been designed with no secret key. Hash functions and truly random sequence generators are of this kind.

2. *Private-key algorithms*, also known as *symmetric-key primitives*, use a private key in their constructions. Private-key algorithms are deployed in symmetric encryption algorithms, keyed hash functions or message authentication codes, pseudo-random sequences and authentication protocols.

3. *Public-key algorithms* or *asymmetric primitives* apply both private and public keys. Asymmetric encryption algorithms, identification primitives, and digital signatures are some applications of public-key cryptography.

Encryption Algorithms

Encryption algorithms, also known as *ciphers*, are generally used to provide confidentiality and secrecy of information. They are key-dependent transformations which map the input data (*plaintext*) to the output data (*ciphertext*). Ciphers are asymmetric or symmetric depending to the type of key used. Asymmetric ciphers use two different keys for encryption and decryption. One key is private and known to only one party, while the other one is public and available to all parties. The keys are mathematically related; however knowing the public key, it is computationally infeasible to derive the private key. Asymmetric ciphers are widely used in authentication and digital signature protocols.

Symmetric ciphers use a private key often referred to as *secret key*. The secret keys for encryption and decryption are identical. All the parties who have access to the secret key can communicate together securely. *Stream ciphers* and *block ciphers* are different categories of symmetric ciphers.

A stream cipher, shown in Figure 1.1a, encrypts individual elements (typically bits) of the plaintext message, one at a time, with the corresponding character of

the key-stream. The idea of stream ciphers was inspired by the Vernam cipher or one-time pad (OTP). OTP combines the plaintext bits with a random key-stream one at a time to generate the ciphertext. However, OTP is not practical as to encrypt a n-bit message string, we need a random key-stream of the same length as the message. A stream cipher applies a pseudo-random key-stream usually generated from a random seed value. The same seed value is used as the key for the decryption of the ciphertext stream.

Block ciphers follow a different approach to symmetric encryption, see Figure 1.1b. They are deterministic algorithms which operate on large and fixed-length blocks of data. Block ciphers are versatile primitives. In addition to encryption, they are used in the construction of pseudo-random number generators, stream ciphers, and hash functions. Block ciphers and hash functions are the main subjects of cryptanalysis in this thesis.

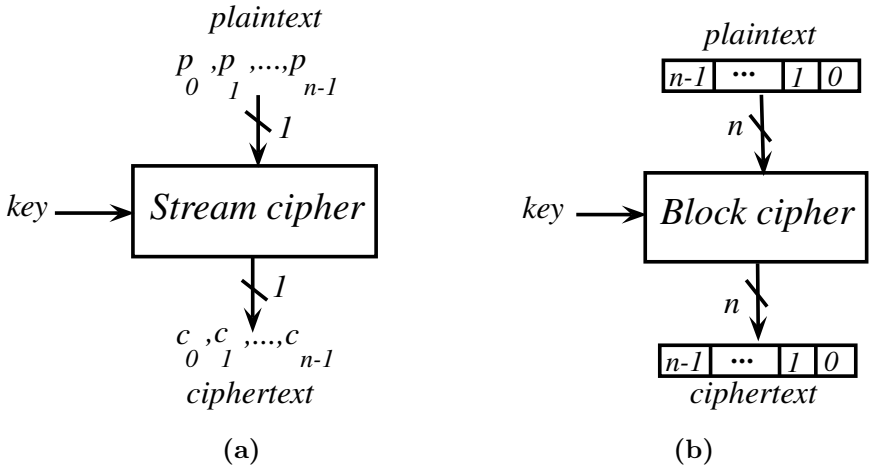


Figure 1.1: n-bit stream and block ciphers

Block Ciphers

Block ciphers serve as the major component in the design of many cryptographic protocols. A block cipher is typically an iterative algorithm and can be viewed as a large key-dependent substitution cipher. It is a bijective transformation and encrypts every plaintext to a unique ciphertext. In a secure cipher, the

ciphertext by itself is supposed to provide as little information about the plaintext as possible. So, it must be infeasible to recover the plaintext with no knowledge about the secret key. However, with the secret key one can easily decrypt a ciphertext, usually by applying the matching decryption algorithm (that runs the inverse operations in the reverse order), and obtain the right plaintext.

An adversary is able to *totally break* a block cipher if it is possible to use a *key-recovery* attack to find the secret key; or *partially break* it by finding part of the plaintext corresponding to a given ciphertext. The attacks on block ciphers with fixed keys are mainly classified into the following three (note the adversary knows about the applied encryption algorithm and wants to recover the key or plaintext):

1. *Ciphertext-only*: The adversary has access only to a set of ciphertexts.

2. *Known-plaintext*: Some plaintext-ciphertext pairs are available for the analysis.

3. *Chosen-plaintext*: The adversary can choose some plaintexts and their corresponding ciphertexts would be available to him.

If a cipher is secure against chosen-plaintext attack, it is also secure against ciphertext-only and known-plaintext attacks [75]. *Chosen-ciphertext* and *related-key* attacks are two additional hypothetical classes of attacks, still interesting in the analysis of ciphers. In the chosen-ciphertext attack the adversary is allowed to decrypt some ciphertexts of his own choice. Security under such attack scenarios is typically needed in application protocols that allow the attacker to actively inject carefully crafted ciphertexts in the network and observe how they are handled. In the related-key attack, the adversary can choose some plaintexts and then have access to the encryption of those plaintexts under two or more unknown keys with certain relationships.

Hash Functions

A *hash function* maps an arbitrary-length message into a fixed-length message digest (a.k.a. *hash value*). It distributes all the input messages evenly among the

possible hash values. Cryptographic hash functions are constructed from one-way compression functions. A one-way compression function transforms a fixed-length input to a fixed-length output, in such a way that only knowing the output the adversary is unable to compute any of the inputs which generate this output. So it is easy to generate the hash value by a cryptographic hash function, but it is intractable to determine the original message from a given digest. Hash functions can be divided into two categories depending on whether or not a secret key is used in their design:

1. *Keyed hash functions* are deterministic functions that take two inputs, a variable-length message and a fixed-length key and generate a fixed-length output. They also may be referred to as *Message Authentication Codes (MAC)*.

2. *Unkeyed hash functions* are deterministic functions that translate an arbitrary message to a fixed-length hash value without using a secret key. Unkeyed hash functions can be generated based on iterative structures.

Hash functions are employed in many different security protocols. Message integrity and data authentication are two important applications of hash functions. Suppose a message is transmitted via an insecure channel. Meanwhile the message digest is transmitted via a secure channel. The recipient calculates the corresponding hash value for the received message and compares it with the received digest. If the message has not been changed during the transmission, the received hash value and the calculated one are equal. One of the prominent applications of hashing is for generation of short digital signatures for authentication purposes. Instead of signing the whole message (which could be very long), the signature is created for its digest obtained by hashing. On the receiving side, the verification of signature progresses in the following two steps: 1) the (long) message is first hashed giving the original digest, 2) the digest obtained from the signature is compared to the original one. If they are equal, the verification is successful. Otherwise, it fails. Another popular application of hash functions is in access control systems for password validation. Instead of storing the plain password, its hash value is stored in the database. Then when a user tries to log

in to the system, the hash of the entered password is calculated and compared with the stored one. The access is granted to the user if both hash values are equal.

Security Analysis of Symmetric Primitives

Together with efficiency, security is an important property of cryptographic primitives. Cryptanalysis is the science of studying security properties of primitives and analysing their weaknesses. Every cryptographic algorithm provides a well-defined collection of security goals. The following list specifies an example collection of some standard goals for block ciphers and cryptographic hashing.

- Block cipher goals:
  - The ciphertext is indistinguishable from a truly random output stream.
  - Guessing the secret key is infeasible. This means the probability of choosing the right key is equal to the probability when the adversary chooses the key randomly and uniformly from the whole key space.
  - Recovering the plaintext from a ciphertext is infeasible.
- Hash function goals:
  - The hash value is indistinguishable from a random output value.
  - It is computationally impossible to invert a hash function (i.e. one-wayness or preimage resistance)
  - It is computationally impossible to find different messages with the same hash values (i.e. collision resistance)

Many different techniques are used to analyse symmetric cryptographic algorithms. One way is to consider the whole algorithm as a black-box and try to solve the problem without any knowledge of the internal design. This analysis is also known as *black-box cryptanalysis*. On the contrary, *statistical* and *algebraic cryptanalysis* exploit some weaknesses in the design of cryptographic primitives. Statistical cryptanalysis (such as linear and differential analysis), studies the building blocks of the algorithm and examines properties of the low-level

Boolean transformations in the design. Algebraic cryptanalysis uses the algebraic representation of the design components to construct a linear/non-linear system of equations on the input/output variables. Cryptanalysis of cryptographic algorithms concentrates on the mathematical structures and tries to explore them to reveal secret elements or break their security properties. It completely ignores the implementation issues of cryptographic algorithms. However, even if a cryptographic algorithm is secure against cryptanalysis, its implementation may leak some information about the secret elements. The security analysis of implementations of cryptographic algorithms is also known as *side channel attacks*.

Motivation and Significance of This Thesis

Cryptanalysis of symmetric algorithms including block ciphers and hash functions is the main subject of this thesis. As mentioned earlier, block ciphers and hash functions are two major primitives which are used in many security protocols to protect sensitive data from disclosure and unwanted access. Moreover, with the advance of technology, there is always a need to design new cryptographic algorithms suitable for the newly available systems and resources. Therefore, it is always a question that whether these algorithms are secure enough to protect information. Cryptanalysts always study crypto-algorithms to make sure they are not vulnerable to the analysis techniques, considering both design and current available resources.

In this thesis, our aim is to contribute in the development of more secure cryptographic designs. We pursue this goal by improving the current well-known cryptanalytical techniques and presenting new methods for the cryptanalysis of symmetric algorithms.

Thesis Contributions

The thesis is outlined as bellow.

Details on the design of block ciphers and hash functions are described in Chapter 2. The cryptanalysis techniques applicable to symmetric designs are also categorised and explained in this chapter.

Chapter 3 presents a new technique for the truncated differential analysis of block ciphers. This analysis benefits from differential distributions of the state symbols. We introduce our proposed method and show its application on the lightweight block cipher LBlock. The key-recovery attacks are applied on LBlock using the truncated differential method and extended by considering the key schedule. The attack is implemented on a small version of LBlock to verify the theoretical results.

Chapter 4 includes two parts. In the first part, the rebound attack on the generalised Feistel-SP networks is described and improved. The presented analysis can be used to generate known-key and chosen-key distinguishers on block ciphers under Feistel-SP designs. Moreover, the main application of this method is on the hashing modes of the Feistel-SP ciphers, or the Feistel-SP hash functions. In the second part, the Camellia block cipher is analysed using the proposed methods. The analysis is optimised to attack the Camellia block cipher. The rebound attack on Camellia is implemented to confirm the results. Finally, collision attacks and distinguishers are also found for the hashing modes of Camellia.

Chapter 5 discusses rebound attacks on the Grøstl hash function and its predecessor Grøstl-0. Grøstl was a dedicated hash function submitted to the SHA-3 competition. The SHA-3 competition was held by NIST to find a new standard secure hash algorithm. The competition was active during the early years of this thesis. The preimage security of Grøstl is examined in this chapter and security bounds are proven for the compression function and the hash function against multi-target preimage attacks. Then multi-target preimage attacks and preimages are found for chosen sets of output values for the Grøstl-0 compression function.

2

Block Ciphers and Hash Functions

The main topic of the thesis is a study of security of both block ciphers and hash functions. This chapter introduces the background information necessary to understand the results obtained. In particular, we introduce different approaches in the design of these primitive as well as their security requirements. Various symmetric cryptanalysis methods are also reviewed in this chapter.

2.1 Block Ciphers

Block ciphers are symmetric cryptographic primitives, which are used to provide confidentiality of information [75]. A secure channel is created between two or more parties by sharing a secret key. Encryption and decryption are applied on the plaintext and ciphertext using the same secret key. Having no knowledge about the secret key, the adversary is not able to obtain the plaintext from the ciphertext. A block cipher is defined formally as follows:

Definition 2.1. *An n-bit block cipher is a mapping* $E : \mathbb{F}_2^n \times \mathbb{F}_2^k \to \mathbb{F}_2^n$ *with block*

size n and key size k, such that for each $K \in \mathbb{F}_2^k$, E_K is an invertible mapping (called the encryption function) from $\mathbb{F}_2^n$ to $\mathbb{F}_2^n$. The inverse mapping E_K^{-1} is the decryption function, such that for each $x \in \mathbb{F}_2^n$, $E_K^{-1}(E_K(x)) = x$.

In 1949, Shannon [96] identified *confusion* and *diffusion* as two properties of a *secure cipher* operation. Confusion is associated with the dependency of the ciphertext on the entire secret key bits. So changing even one bit of the key should change the ciphertext. Diffusion refers to making a complex relationship between the plaintext and the ciphertext. More precisely, a slight change in either key or plaintext should change the ciphertext significantly. For example, flipping one bit of the key or plaintext should result in flipping half the ciphertext bits on average. This property is referred to as the *avalanche effect* [32]. If a block cipher (or even a hash function) does not exhibit a substantial avalanche effect, it is easier to break the algorithm due to the poor randomisation. In block ciphers, the designers improve the avalanche effect by providing confusion and diffusion. Confusion and diffusion are typically implemented by a series of non-linear transformations (such as substitutions and modular addition), and linear mappings (such as permutations), respectively.

The modern block ciphers are mostly *iterated block ciphers* in which there is a single transformation called the *round function*. The round function is iterated many times. For instance, the well-known DES algorithm [34] iterates the round function 16 times. Usually every round uses a different sub-key derived from the secret key. To make the cipher invertible, the round function is a bijection on the round input for each value of the sub-key. Assume an n-bit iterated block cipher that iterates a round function f for r rounds. The round sub-keys $K_1, K_2, ..., K_r$ are generated from the original secret key through the key scheduling process. Figure 2.1 shows the structure of this iterated block cipher.

There are many different approaches in designing iterated cryptographic algorithms. The following three are the most prominent: *substitution-permutation networks (SPN)* [96], *Feistel networks* [32], and *addition-rotation-XOR (ARX) networks* [97] [1]. Note that a specific design could be a mixture of the three. For instance, a Feistel network may use an ARX round function.

[1] These principles are also used in the design of hash functions

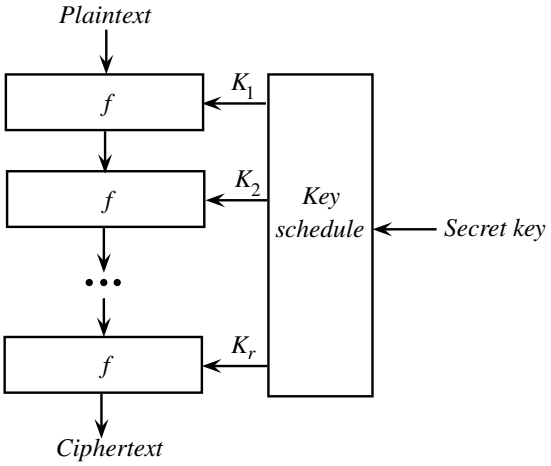


Figure 2.1: Iterated block cipher structure

2.1.1 Substitution-Permutation Networks

The idea of iterated block ciphers based on SPN was formulated by Shannon [96]. The round function of SPN consists of a *substitution layer* followed by a *permutation layer*. The substitution layer is a non-linear stage which is generated from small sized substitution boxes (*S-boxes*). A S-box substitutes a small block of data with another block, and to be secure, must exhibit the avalanche property. To ensure the invertibility of the round function, as well as using the inverse algorithm for the decryption, the S-boxes must be one-to-one mappings. The permutation layer is a linear transformation and as a result should at least permute all the bits in the state block. It gets the output of the substitution layer as the input *state* and by permuting them feeds them into the different S-boxes in the next round. The round sub-key is combined with the input of each round using some group operations, such as XOR or modular addition. Figure 2.2 presents a sketch of a sample SPN cipher.

To explain the SPN structure, we review the round function of two well-known block ciphers AES [26] and PRESENT [19]. The Advanced Encryption Standard (AES) is a standard 128-bit block cipher designed based on the SPN structure. The round function gets the 128-bit input block as a 4×4 matrix of bytes. In the substitution layer, it uses an 8-bit S-box to substitute each byte. The linear transformation creating the permutation layer consists of shifts, and a mixing

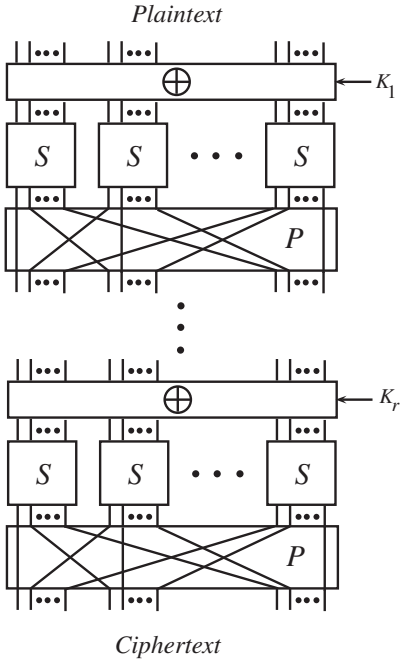


Figure 2.2: Sample SPN cipher

operation which combines the 4 bytes in each column of the state. Then a simple XOR addition combines the state with the round sub-key.

The other example is PRESENT, a 64-bit lightweight block cipher with a simple SPN structure. The round function firstly combines the 64-bit input state with the round sub-key by the XOR addition. Then 4-bit identical S-boxes are applied on every 4 bits, to generate the substitution layer. The permutation layer is a bitwise permutation which is performed on the whole 64 bits.

2.1.2 Feistel Networks

Feistel network (a.k.a Feistel ciphers) [32] is a symmetric structure used in the construction of block ciphers and hash functions, named after its designer Horst Feistel. The Feistel structure splits the block of data into two equal-sized halves. Note, a modified structure of the Feistel cipher called *Unbalanced Feistel cipher*, when the left and right-hand halves are not of equal length.

In a balanced Feistel cipher, the round function is applied to one half using the round sub-key. Then the output of the round function is combined with the

other half. Later the two halves are swapped to make the input of the next round. Encryption and decryption in Feistel ciphers is usually identical, just the reversed order of the round sub-keys is used for the decryption. Hence, the round function does not have to be invertible.

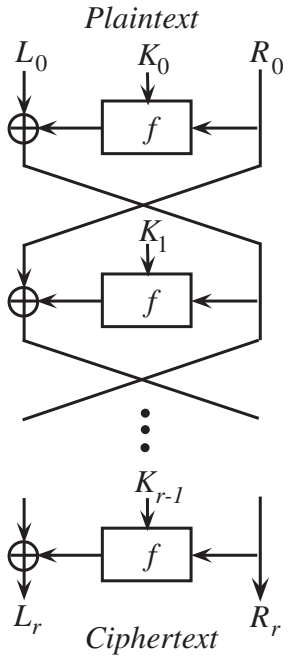


Figure 2.3: A basic Feistel cipher

Figure 2.3 shows the basic structure of a Feistel cipher. Suppose f is the round function, r is the number of rounds, and $K_0, K_2, ..., K_{r-1}$ are the round sub-keys. The input plaintext is split into two equal-sized halves, (L_0, R_0). Then the following pseudo-code shows the basic operations of the cipher:

for $i = 0, 1, ..., r-1$ **do**
  $L_{i+1} = R_i$
  $R_{i+1} = L_i \oplus f(R_i, K_i)$
end for
return (R_r, L_r)

The Data Encryption Standard (DES) [34] is a 64-bit block cipher based on the Feistel structure. The round function in DES, mixes the sub-key with the input state using the XOR addition, then it is followed by a substitution-permutation

structure. DES was the encryption standard from 1975 until the late 90s when it became obvious that it can be broken by an exhaustive attack. Note that this was the result of a short 56-bit secret key. DES was also extensively analysed. Although it was theoretically broken using differential and linear cryptanalysis, its design is still considered as cryptographically sound.

2.1.3 Addition-Rotation-XOR Algorithms

A new design paradigm for symmetric primitives such as, block ciphers and hash functions, uses three operations: modular addition, rotation and XOR addition [97, 106]. The crypto-systems built from these operations are called ARX structures. The round function in an ARX cipher is a combination of these three operations. These ciphers are very fast and cheap in hardware and software. They are immune to side-channel attacks which exploit the run time, such as timing attacks. However, their security against linear and differential cryptanalysis is still an open question. Threefish is a block cipher with ARX-based round functions, and is the basic structure of the Skein hash function [33]. There are similar designs to ARX with small alterations. For example TEA [107], and its extended versions, are ARX-like block ciphers which use shifts instead of rotations.

2.1.4 Modes of Operation

A block cipher encrypts a single fixed-size block of data. To encrypt an arbitrary-length message by a block cipher, different techniques are used, called *modes of operation*. The four most common modes are summarized bellow [75]:

- *Electronic CodeBook (ECB)*: The variable-length message is partitioned into separate blocks, and encrypted or decrypted separately. The last block may be padded to get the right block size. However, ECB mode is not a secure mode of operation. Because identical plaintext blocks always generate the same ciphertexts under the same key. Also re-ordering the ciphertext blocks results in re-ordered corresponding plaintexts after decryption.

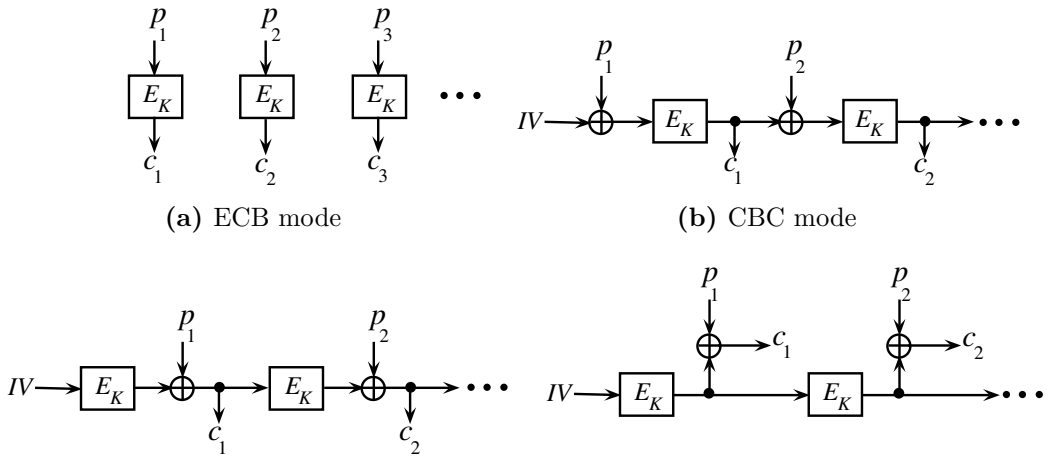
- *Cipher-Block Chaining (CBC)*: This mode uses an initialisation vector (IV). The first plaintext block is XORed with IV to make the input block for the

encryption. Later, to encrypt the next plaintext block, ciphertext of the previous block is used as the new initialisation vector.

- *Cipher FeedBack(CFB)*: The previous ciphertext is encrypted and then XORed with the plaintext block to make the current ciphertext block. An IV is encrypted and used for the first message block encryption.

- *Output FeedBack (OFB)*: The IV gets encrypted repeatedly and then XORed with the plaintext block to create the ciphertext. OFB mode is similar to CFB mode except the value XORed with the plaintext block is not coming from the previous ciphertext. There is no need to keep the IV private, however it must be changed by re-using the secret key.

These four modes of operations are shown in Figure 2.4. Whereas, E_K is the block cipher under the secret key K, p_i and c_i's are respectively plaintext and ciphertext blocks, and IV is the initial vector.



(a) ECB mode **(b)** CBC mode

(c) CFB mode **(d)** OFB mode

Figure 2.4: Operation modes in block ciphers

2.1.5 Security Analysis

It has been always believed that the security of a block cipher relies on the secrecy of the key. If an adversary breaks a block cipher and recovers the secret key, he

is able to obtain the plaintext from a corresponding ciphertext, and vise versa. *Key-recovery attack* is usually the result of a chosen-plaintext analysis. A block cipher is considered unbroken if all known attacks have time/data complexity larger than the exhaustive-key search. Nevertheless, the security against key recovery attack is not sufficient for the security of a block cipher. Sometimes discovering weaknesses in a cipher design might end up obtaining the whole or part of the plaintext from a ciphertext.

Another type of cryptanalysis on block ciphers is the *related-key attack* [10]. In a related-key attack, the adversary observes the cipher operation under several different keys with known relations, and then recovers the keys. Although, a cryptographer may never encrypt plaintexts under some related keys; in modern protocols the keys are generated by computers which makes the related-key attack practical. The related-key attack is described in detail in Section 2.3.7.

In addition to the key-recovery attack, an adversary might be able to analyse a cipher against *distinguishing* attacks [40]. Distinguishers are one of the weakest attacks on block ciphers, however they analyse the randomness of the ciphers. An adversary is able to distinguish a block cipher from a random oracle by querying both the cipher and the random oracle.

2.2 Hash Functions

A cryptographic hash function is a one-way function employed in many modern crypto-systems [75]. Mapping arbitrary-length strings to fixed-length strings, unkeyed hash functions are used for data integrity, digital signatures and password protections. There is no secret key in the unkeyed hash functions, and they generate the hash values just by taking a message as the input. Unkeyed hash functions are of particular interest to this thesis, (from now on hash functions will refer to unkeyed hash functions).

Definition 2.2. *A hash function is a mapping* $H : \{0,1\}^* \rightarrow \{0,1\}^n$ *with n-bit hash value, such that the inverting process is not easy to achieve.*

2.2.1 Security Requirements

In the applications of cryptographic hash functions, the hash value is typically a fingerprint of the message and guarantees its uniqueness. This is to say that each message is assigned to a unique hash value. A hash function distributes all finite-length inputs evenly over the hash values. One-wayness is an important property of hash functions and indicates computational hardness of the inverse mapping. A message that hashes to a given hash value, is called the *preimage* of that hash value. After all, as a many-to-one function, mapping different messages (i.e. *collisions*) to the same hash value is inevitable. However, finding collisions must be computationally infeasible. The security requirements for a cryptographic hash function H are listed below [75]:

- *Preimage resistance*: for every hash value y, it is computationally infeasible to find a message x such that $H(x) = y$.
- 2^{nd}*-Preimage resistance*: for every pre-specified message x, it is computationally infeasible to find another input message x' such that $x \neq x'$ and $H(x) = H(x')$.
- *Collision resistance*: it is computationally infeasible to find any two distinct input messages $x_1 \neq x_2$ which hash to the same value, $H(x_1) = H(x_2)$.

When we say a calculation is computationally infeasible, it is impossible to achieve using the current algorithms and computing resources. Note, for an n-bit hash algorithm, the generic attacks find preimage and 2^{nd}-preimage in 2^n steps, while they find collisions after $2^{n/2}$ steps (normally a step is equivalent to a single evaluation of the hash function). Therefore, a hash function is secure if all the known attacks need more computations than the generic attacks.

2.2.2 Hash Function Construction

The majority of cryptographic hash functions use an iterative process that applies a *compression function*. The idea was invented independently by Merkle [77] and Damgård [30], known as *Merkle-Damgård construction* (illustrated in Figure 2.5).

A compression function f maps a longer fixed-size input to a shorter fixed-size output:

$$f : \{0,1\}^{l+r} \rightarrow \{0,1\}^{l}$$

The input message is padded and then divided into r-bit message blocks. The padding process adds extra bits to make the message length a multiple of r. Sometimes a length-block is also added to show the bit-length of the unpadded message. The compression function generates an l-bit result, called the *chaining value*, by taking an r-bit message block and the l-bit compression result of the previous message block. An *initial value* (IV) is used for the compression of the first message block. At the end, an optional final transformation g is applied on the last chaining value to generate the n-bit hash value. When n is equal to l, g is often the identity mapping. Assume the input message M is partitioned into t message blocks after the padding. The following is the formal description of the n-bit iterated hash function H:

$$H(M) = g(h_t)$$

$$where,\ h_0 = IV;\ h_i = f(h_{i-1}, m_i), 1 < i \leq t$$

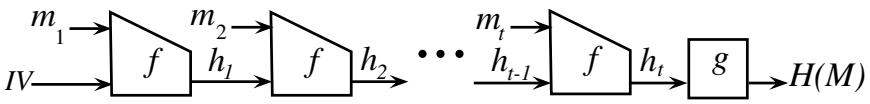


Figure 2.5: Merkle-Damgård construction

In the Merkle-Damgård construction, if the padding includes the length-block, a collision-resistant compression function results in a collision-resistant hash function [30, 77]. Adding such a length-block in the padding process is called *Merkle-Damgård strengthening*.

In 2007, *sponge construction* was proposed to design hash functions and stream ciphers. The sponge construction [8] is a class of algorithms with finite internal state which produces an arbitrary length output bit stream from an input bit stream of any length. As Figure 2.6 shows the sponge structure consists of two phases, the *absorbing phase* which is followed by the *squeezing*

phase. First of all, the message is padded to generate r-bit message blocks. then the absorbing phase takes the message blocks one-by-one and after XORing with the first r-bit of the state applies the transformation function f on the state. After absorbing the whole message blocks, it is the squeezing phase which extracts the first r bits of the state as the output block and applies the transformation function f to generate the new state. The number of iterations for the squeezing phase depends on the length of the output. Finally the output is truncated to generate the desired output length.

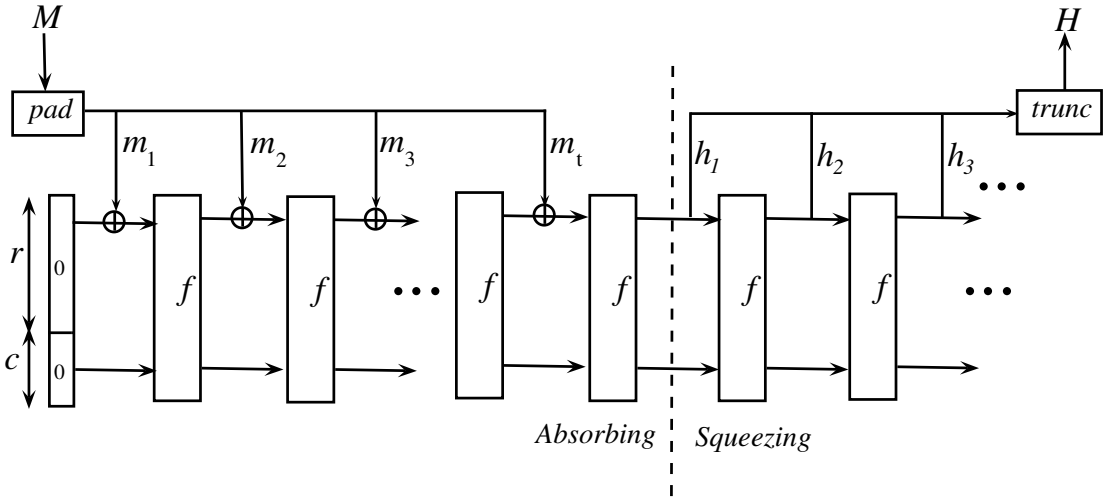


Figure 2.6: The sponge construction

Recently, the sponge construction has been used to design several hash functions. From them, Keccack [9] is the winner of the SHA-3 competition and has been selected as the standard hash function.

Moreover, hash functions can be classified into three following categories based on their design structures:

- *Block-cipher based hash functions*: A block cipher is used to build the compression function.

- *Dedicated hash functions*: A dedicated compression function is designed for a hash function. The compression function might be based on a simplified block cipher. MD- and SHA-family of hash functions such as, MD4 [89],

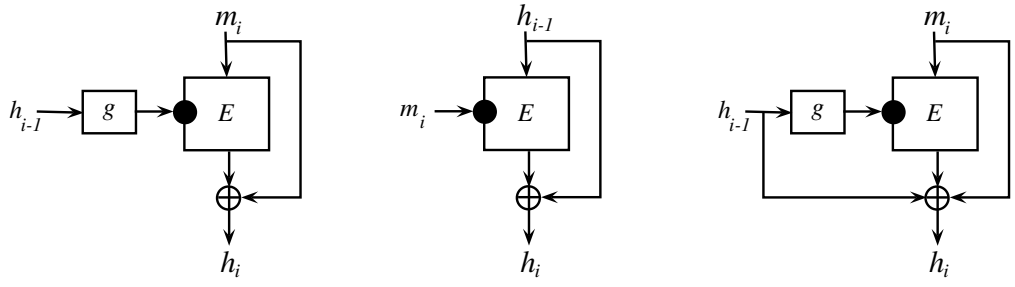
MD5 [90], SHA-1 [81] and SHA-2 [82] are some examples of dedicated hash functions.

- *Hash functions based on intractable Problems*: They are based on difficult problems such as exponentiation, squaring, knapsack and discrete logarithm [86].

2.2.3 Block-Cipher Based Hash Functions

Block ciphers can be used as the building blocks of hash functions, by serving as the compression functions. There are many different ways to build a compression function from a block cipher. Three most common schemes are, *Matyas-Meyer-Oseas*, *Davies-Meyer*, and *Miyaguchi-Preneel* [75]. In these schemes, an n-bit block cipher E is used to build an n-bit chaining value h_i. Also a feed-forward is mixed with the output value to make the hash function irreversible and preimage resistant.

Figure 2.7 shows the structure of these three schemes. In the Matyas-Meyer-Oseas and Miyaguchi-Preneel schemes, a function g is used to map the previous chaining value to a key-sized value. This value serves as the key in the encryption of the current block. In the Davies-Meyer scheme the message block is used as the key to encrypt the previous chaining value.



(a) Matyas-Meyer-Oseas **(b)** Davies-Meyer **(c)** Miyaguchi-Preneel

Figure 2.7: Block-cipher based compression function schemes

2.3 Cryptanalysis Methods

Cryptanalysis refers to study of cryptographic primitives by evaluating their security properties. A cryptanalysis usually exploits the weaknesses found in a primitive design. For example, an attacker might apply a key-recovery attack on a block cipher by linear analysis. In this section we explain some of the common techniques for the analysis of block ciphers and hash functions.

2.3.1 Black-Box Analysis

A black-box analysis is an attack that works independently of the crypto-algorithm internal structure. They use high-level information about a primitive behaviour such as, the block length and hash size in block ciphers and hash functions, respectively. Black-box attacks provide upper bounds for the complexity of other cryptanalysis techniques.

The most common black-box analysis is the *brute-force search* [75]. In the case of block ciphers the adversary exhaustively tests all the possible keys on a small number of plaintext-ciphertext pairs to obtain the right key. For example if the key size in a block cipher is k bits, the brute-force attack (a.k.a. *exhaustive-key search*) is on the order of 2^{k-1} on average. For measuring the complexity we say the attack takes 2^{k-1} encryption calculations. Testing just one key, the probability of picking the right key is 2^{-k}.

The brute-force attack is also used to find preimages and 2^{nd}-preimages for hash functions. Given an n-bit hash function, the attacker should search 2^n messages to find the preimage of a randomly given hash value. To apply collision attacks on hash functions, a better black-box method can be used. The method applies the *birthday paradox* [101]. The adversary tests random messages and stores their hash values until he finds two messages with the same hash value. According to the birthday paradox, for an n-bit random hash function, the first collision is expected to be found after $\sqrt{\frac{\pi}{2}2^n}$ tests [101], which is approximately $2^{n/2}$ hash computations.

2.3.2 Algebraic Analysis

An adversary launches an algebraic attack on a primitive by constructing a system of multivariate (typically non-linear) equations from input and output states (and the key in block ciphers). Then he recovers the unknown variables by solving the system of equations. For example, he may perform a key-recovery attack on a block cipher, or a preimage attack on a hash function.

The efficiency of an algebraic attack is defined by the complexity of solving the system of equations. There are different methods to solve a system of non-linear equations. One way is linearisation of the system, which is performed by assigning an independent variable to each non-linear monomial. Linearisation might fail when there are linear dependencies between polynomials. In this case the extended versions, XL [22] (i.e. eXtended Linearisation) and XSL [23] (i.e. eXtended Sparse Linearisation) might successfully solve the system. Another way to solve a system of equations is based on the computation of Gröbner basis [20].

2.3.3 Linear Analysis

Linear cryptanalysis was discovered by Matsui in 1992 to analyse block ciphers and led to the cryptanalysis of DES [69, 70, 71]. This cryptanalysis exploits linear expressions of the input and output bits that occur with very high or very low probability [43]. Assume X and Y are respectively the input and output vectors, also A and B are the input and output linear masks. Let p be the probability that the following expression holds in a selected primitive.

$$A.X \oplus B.Y = 0$$

In an ideal primitive the above expression holds with probability $1/2$. The deviation of the probability p from the random probability $1/2$ is called the *linear probability bias*, i.e. $\varepsilon = |p - 1/2|$. The linear cryptanalysis is more successful when using an expression with a higher bias.

High probability linear expressions are generated by considering the properties of non-linear components, such as S-boxes. When a linear approximation is developed for a round function, it is possible to concatenate the approximations

of consecutive rounds together and generate the linear approximation for some number of rounds, also referred to as *linear characteristic*. Matsui introduced the *piling-up lemma* [70], to construct linear characteristics for a crypto-algorithm. According to the piling-up lemma, for n independent random binary variables $X_1, X_2, ..., X_n$ the probability of the linear approximation generated from concatenating these variables is:

$$\mathbf{P}(X_1 \oplus X_2 \oplus ... \oplus X_n) = 1/2 + 2^{n-1} \prod_{i=1}^{n} \varepsilon_i$$

where X_i represents a linear approximation for round i and ε_i is its probability bias.

Linear analysis is used for key-recovery and distinguishing attacks on block ciphers. The adversary uses the piling-up lemma to generate a highly biased linear approximation involving only plaintext and the last round input bits of an iterated block cipher. Then for sample plaintext-ciphertext pairs he guesses the last-round partial sub-key bits and decrypts the ciphertexts through the final round. A counter is assigned to each guessed partial sub-key and incremented when decryption of a ciphertext holds true for the linear expression. The guessed partial sub-key with higher bias (i.e. its counter deviates larger from half of the samples) is assumed as the right partial sub-key. This process is continued to recover all the key bits by decrypting more rounds or checking other linear approximations. To examine a linear expression with bias ε, the attacker requires a small multiple of ε^{-2} known plaintext-ciphertext pairs.

In the case of hash functions, linear analysis might be employed for distinguishing attacks. However, the usage of non-linear components in the design of hash functions usually makes it improbable to find a highly-biased linear expression.

2.3.4 Differential Analysis

Differential analysis is a powerful technique which is used for the analysis of block ciphers and hash functions. It basically studies the affect of input differences to the output differences and exploits the highly probable difference propagations

through the round function. The differential analysis was discovered in the academic society by Biham and Shamir in 1990 [11]. However, it was revealed later that IBM and NSA had been aware of this method since 1974 [21].

The difference usually refers to the XOR difference between a pair. Assume X and X' are two input vectors to a primitive and Y and Y' are the corresponding outputs. The pair of differences $(\Delta X, \Delta Y)$ is called a *differential*, where $\Delta X = X \oplus X'$ and $\Delta Y = Y \oplus Y'$. In an n-bit random permutation, a specific output difference ΔY occurs from a given input difference ΔX with probability $1/2^n$. The differential cryptanalysis takes the advantage of differentials with much higher probabilities than the random case. For an iterated structure, the adversary examines a sequence of high probability differentials through the rounds, referred to as a *differential characteristic* (a.k.a. *differential trail*) . Assume a differential trail is a sequence of differentials for r rounds, and the differential probability for round i is p_i, the differential trail probability $\mathbf{P}_D$ is

$$\mathbf{P}_D = \prod_{i=1}^{r} p_i$$

A differential trail with a relatively higher probability than random, can be used to distinguish a primitive from a random permutation.

Linear components, such as XOR addition or rotation, are deterministic transformations. They transform a given difference to a specific output difference with probability one. On the contrary, a non-linear component (e.g. S-box) is a probabilistic transformation. To find the differential probabilities a *Difference Distribution Table (DDT)* is constructed, which for every given input difference represents the number of occurrences of every output difference. Probability of an output difference occurring from a given input difference is calculated from the DDT.

Differential cryptanalysis is used for key-recovery on block ciphers under a chosen-plaintext attack. As shown in Figure 2.8, firstly the adversary discovers a high probable differential trail for $r-1$ rounds of a block cipher. Where ΔP is the plaintext difference and ΔV is the input difference of the last round. Then partially/fully decrypts the ciphertext pairs C and C' through the last round, and calculates the probability of the desired output difference ΔV, for every candidate

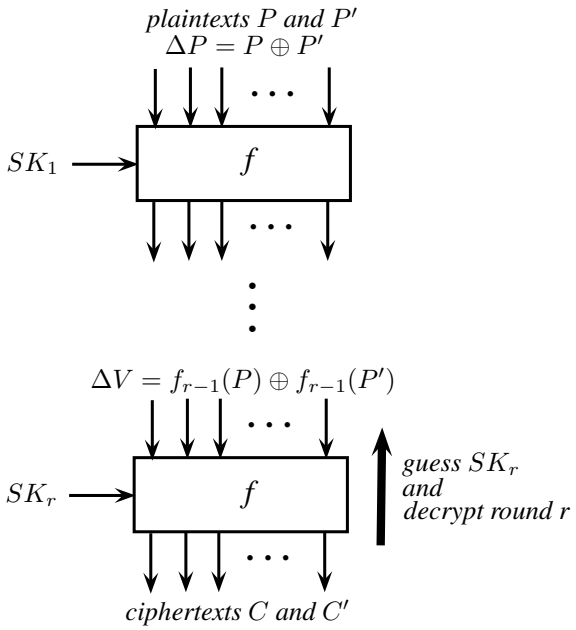


Figure 2.8: Differential attack on an iterated block cipher

of the final round sub-key SK_r. Candidate sub-key with higher probability and suitably close to the expected probability is taken as the right sub-key. Wrong sub-key guesses are expected to act as a random guess and achieve the correct difference with a very low probability [43, 54]. The number of plaintext pairs (or data) required to mount the attack is proportional to the inverse of the differential trail probability. If a differential trail for $r-1$ rounds occurs with probability p_D, then c/p_D plaintext pairs are required for the key-recovery attack (where c is a small constant) [95].

We use differential cryptanalysis and its extended methods to analyse some symmetric primitives in this thesis.

2.3.5 Truncated Differential Analysis

Truncated differential analysis [51] was presented by Knudsen as a generalisation of differential analysis. Unlike the classical differential attack, a truncated differential does not follow a specific difference between two input plaintexts. Instead, it considers partially determined differences. For example, a truncated differential

may predict the differences of only some bits not the full block.

Knudsen explained the truncated differential analysis more clearly by an example on a toy cipher called CipherFOUR [53]. CipherFOUR is a 16-bit iterated SPN-cipher with r rounds. The round function includes four identical 4-bit S-boxes followed by a bit-wise permutation as shown in Figure 2.9. The S-box transformation is also shown in Table 2.1.

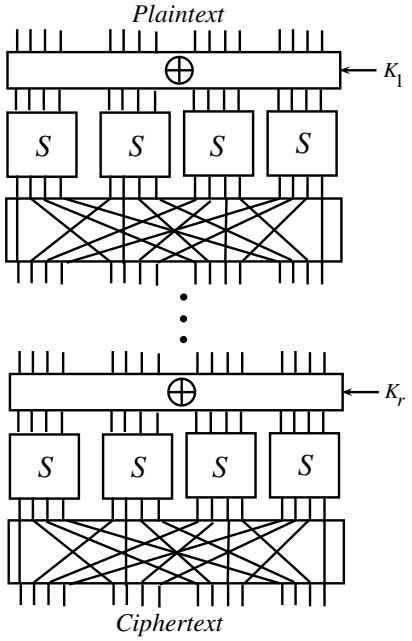


Figure 2.9: CipherFOUR encryption [53]

Table 2.1: CipherFOUR S-box transformation [53]

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

Assume the input difference to the S-box transformation is 0010, the output difference is either 1, 2, 9 or A. So, the output difference has the form of $*0**$ with the probability 1. It means we just know the difference of one bit while the others are not clear.

Now if the input difference to the whole state is 0000 0000 0010 0000 after one round of encryption the output difference is either 0000 0000 0010 0000,

0000 0000 0000 0010, 0010 0000 0010 0000 or 0010 0000 0000 0010. After another round of encryption the output difference has the form of $*0** \ 0000 \ *0** \ *0**$ with probability 1.

Similar to the classical differential analysis, we may follow a truncated differential trail with a probability less than 1, then use the same technique to recover the secret key.

2.3.6 Other Differential Analysis Extensions

Differential analysis is extended to several cryptanalysis methods for analysing symmetric primitives. Bellow are some of the extended differential techniques.

Linear-Differential [58] was introduced by Langford and Hellman as a combination of linear and differential analyses and works based on linear-differential probabilities. The adversary constructs a highly probable differential trail for part of the primitive, then generates a largely-biased linear approximation for the following rounds respecting the output pairs.

Higher-Order Differential [51] is a higher-order variant based on differences of differences and has an algebraic order. An order d derivative of a function at d points involves 2^d function values. For example, 2^{nd} order derivative of function f at points α and β is

$$\Delta_{\alpha,\beta} f(x) = f(x) \oplus f(x \oplus \alpha) \oplus f(x \oplus \beta) \oplus f(x \oplus \alpha \oplus \beta)$$

The conventional differential is a 1^{st} order differential. Integral analysis (a.k.a. square attack) [29] and boomerang attack [104] are some variants of higher order differentials.

Impossible Differential [12, 13] exploits differentials with zero (or very low) probabilities. The impossible differential might happen in an intermediate state of a trail. The idea is based on finding an impossible differential which is highly probable in a random permutation. For example the output state difference contains a few bits with non-zero differences. In the key-recovery attacks, the impossible differential is easily used to eliminate the wrong keys and narrows down the candidate keys.

Multiple Differential [16] considers a set of differentials. Multiple differential is more general than truncated differential because the set of output differences differs for various input differences. Calculation of data and time complexity depends on the differential structures and the statistical techniques used to rank the keys. The input difference structures are typically selected in a way that more samples are generated from less plaintexts.

2.3.7 Related-Key Attack

In 1993, Biham introduced *related-key attack* to analyse block ciphers [10]. The related-key attack violates the key scheduling of a block cipher and shows how small details can affect security of the cipher. Biham described that the related-key attack can be used to apply chosen-key attacks on block ciphers in which the attacker only knows the relation between different sets of keys and does not know the keys.

Usually, the related-key attack is used as a variation of differential analysis which applies the *related-key differential attack* [49]. These attacks accept keys and plaintexts with specific differences, then investigate the ciphertexts to recover the key. For example, assume an iterative cipher E with r rounds. This cipher encrypts a pair of plaintexts with difference ΔP to a pair of values with difference ΔV after application of $r-1$ rounds, while a pair of keys is used for the encryption with the difference of ΔK, i.e. $E_{\Delta K}(\Delta P) = \Delta V$ after $r-1$ rounds. Now the adversary chooses the plaintexts with the specified difference and encrypts them for r rounds under the pair of unknown keys, whereas they have the desired difference. Then, she searches through the possible sub-keys of the last round to decrypt the ciphertexts while they generate the values with the difference Δ_V after $r-1$ rounds. Now she recovers the sub-keys for the last round and can continue to recover the whole keys.

There are several security protocols which needs their underlying block cipher to be secure against relater-key attacks. For example, 3GPP (3rd Generation Partnership Project) is based on confidentiality and integrity schemes, $f8$ and $f9$ [102], that both are based on KASUMI [103] block cipher. The security

of these schemes depends on the security of KASUMI against related-key attacks [47]. Moreover, McOE-X a varient of McOE, a family of almost foolproof on-line authenticated encryption schemes, requires the security of its underlying block cipher against related-key attacks [35].

As well as block ciphers, the related-key analysis can be used to attack hash functions [49]. They can even be used to apply collision attacks on hash functions. Assume a hash function is designed based on a block cipher under the Davies-Meyer scheme. Therefore the message is considered as the key for the underlying block cipher. In this case, applying the related-key attack on the block cipher, succeeds to find a pair of messages which are generating the same hash value.

3

Truncated Differential Analysis of LBlock

Recently the differential cryptanalysis based on probability distribution has attracted a lot of attention in the analysis of block ciphers. These type of attacks typically require a lower amount of data in comparison to the classical differential attacks. In this chapter we present a truncated differential analysis of LBlock and investigate difference distributions of the state nibbles. After finding a distribution that significantly differs from that of a random permutation, we use log-likelihood ratio (LLR) statistical test to construct a distinguisher. We apply the key-recovery attack by exploiting the key-schedule, and concatenating additional rounds to the beginning and end of the distinguisher. To validate the analysis we implement the attack on LBlock reduced to 12 rounds. Then single key and related key attacks are applied to 18 and 21 rounds of LBlock, respectively.

This chapter is structured as follows. Lightweight block cipher LBlock is described in Section 3.1. Likelihood ratio test which is our distinguishing technique is explained in Section 3.2. A framework to apply the key-recovery attack using

the truncated differential distribution, while benefiting the key schedule properties is introduced in Section 3.3. Section 3.4 discusses the complexity of the attack and includes the empirical results. Section 3.5 presents a single-key attack on 18 rounds as well as related-key attacks on 20 and 21 rounds of LBlock. Finally, we conclude this chapter in Section 3.6.

3.1 Lightweight Block Cipher LBlock

With the advent of RFID technology in communication applications, traditional block ciphers are generally not suitable for resource constrained devices. Lightweight block ciphers (with smaller block and key size) are a new class of ciphers designed for such environments. Recently there have been a lot of new lightweight designs, examples include: HIGHT [44], PRESENT [19], PRINTcipher [50], and LBlock [110].

3.1.1 LBlock Description

LBlock [110] is a lightweight block cipher with a block size of 64 bits and a key size of 80 bits. The design is a 32-round balanced Feistel where the input block is divided into two 32-bit halves, denoted the *left-hand* half (most significant bits) and the *right-hand* half (least significant bits).

The structure of LBlock is depicted in Figure 3.1. Since all the state functions operate on 4 bits, it is convenient to represent the state as a sequence of nibbles using the following notation $\boldsymbol{x} = (x_{15}, \ldots, x_1, x_0)$.

Each round includes a key addition, where the round sub-keys are 32-bit values denoted by $SK[i]$. The round function F consists of a XOR key addition, a non-linear S-box layer, and a linear permutation layer. The S-box layer applies 8 different S-boxes (s_i) in parallel. The linear permutation layer simply re-orders the 8 nibbles in the state. The round function is shown in Figure 3.2.

LBlock uses a key scheduling function to expand the 80 bit master key K into 32 round sub-keys $SK[i]$, each being 32 bits in size. The master key K is stored in a register, denoted by the sequence of bits $k_{79}k_{78}k_{77}k_{76}\ldots k_1k_0$. The key register K is updated by the scheduling process and every round the 32 most significant

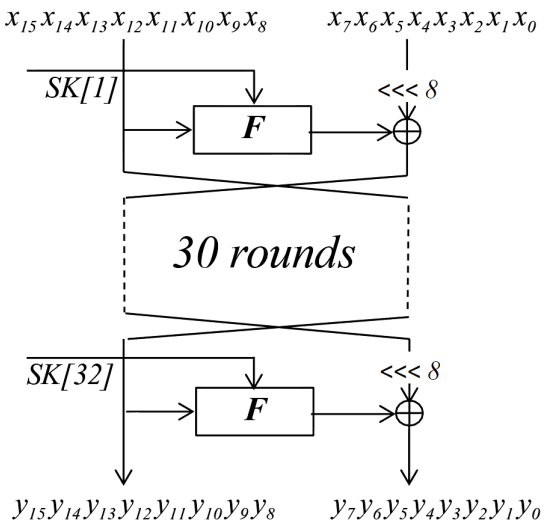


Figure 3.1: LBlock Feistel structure

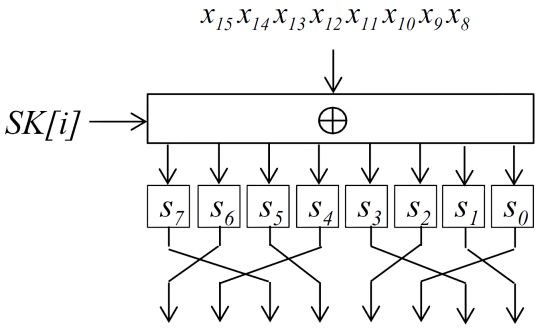


Figure 3.2: The round function of LBlock

bits of the current register become the round sub-key. The key scheduling process is as follows:

$SK[1] = [k_{79}k_{78}...k_{48}]$

for $i = 1, 2, ..., 31$ **do**

1. $K \lll 29$
2. $[k_{79}k_{78}k_{77}k_{76}] = s_9[k_{79}k_{78}k_{77}k_{76}]$ and $[k_{75}k_{74}k_{73}k_{72}] = s_8[k_{75}k_{74}k_{73}k_{72}]$
3. $[k_{50}k_{49}k_{48}k_{47}k_{46}] \oplus [i]_2$
4. $SK[i+1] = [k_{79}k_{78}...k_{48}]$

end for

where s_8 and s_9 are two 4-bit S-boxes and $[i]_2$ is the binary representation of i.

3.1.2 Other Cryptanalyses of LBlock

Similarly to the other lightweight block ciphers, LBlock has attracted a significant amount of cryptanalysis. For instance, related-key impossible differential attacks were successfully applied to 21 and 22-round LBlock [63, 79]. A 16-round related-key truncated differential is exploited to launch an attack on 22-round LBlock [62]. In [92], a 15-round distinguisher is proposed, allowing an integral attack for up to 22 rounds. Zero-correlation linear cryptanalysis of 22-round LBlock is presented in [99]. All the attacks published so far require high amount of memory and data. A comparison between prior cyptanalyses complexities and our results is given in Table 3.1.

Table 3.1: Attacks on LBlock

Type of Attack	rounds	Data	Time	Reference
Related-key impossible differential	22	2^{68}	2^{70}	[79]
Related-key differential	22	$2^{64.1}$	2^{67}	[62]
Integral	18	$2^{62} + 2^{20}$ memory	2^{36}	[110]
Integral	22	$2^{61} + 2^{63}$ memory	2^{70}	[92]
Zero-correlation linear	22	$2^{60} + 2^{64}$ memory	2^{79}	[99]
Truncated differential	18	2^{23}	$2^{68.38}$	Section 3.5.1
Related-key truncated differential	20	2^{27}	$2^{74.33}$	Section 3.5.2
Related-key truncated differential	21	2^{30}	$2^{77.89}$	Section 3.5.2

3.2 Likelihood Ratio Test

Likelihood ratio test is a statistical test used to compare two distributions and determines the one which is a special case of the other one. This test also examine empirical data and decides which distribution the data is following. In likelihood theory, Kullback-Leibler divergence (or relative entropy) is a measure that quantifies the difference between two distributions [6, 24].

Definition 3.1. *Let* $P = (p_0, p_1, \ldots, p_n)$ *and* $Q = (q_0, q_1, \ldots, q_n)$ *denote two discrete probability distributions of random variables* X *and* Y*, respectively. The*

Kullback-Leibler (KL) divergence between P and Q is defined as follows:

$$D(P||Q) = \sum_{i=0}^{n} p_i \cdot ln(\frac{p_i}{q_i}) \tag{3.1}$$

As in [24], we use the convention that $0 \cdot log\frac{0}{q} = 0$ and $p \cdot log\frac{p}{0} = \infty$.

In the *binary hypothesis testing problem*, one is given a set of empirical data $x = (x_0, x_1, \ldots, x_n)$ taken from N samples. The empirical probability distribution is equal to $\hat{P} = (\hat{p}_0, \hat{p}_1, \ldots, \hat{p}_n) = 1/N \cdot (x_0, x_1, \ldots, x_n)$. According to the *Neyman-Pearson Lemma*, the log-likelihood ratio is the optimal method for determining if the sample data belongs to one of two different probability distributions P or Q [24, 42].

Definition 3.2. *The log-likelihood ratio (LLR) is defined as*

$$LLR(\hat{P}, P, Q) = N \sum_{i=0}^{n} \hat{p}_i \cdot ln(\frac{p_i}{q_i}) \tag{3.2}$$

If $LLR(\hat{P}, P, Q) \geq \Theta$ (Θ is a threshold parameter), the empirical data is accepted as a sample from the distribution P (rejecting Q as the hypothesis). Otherwise, P is rejected in favour of Q. In our analysis, we use this to distinguish between distributions representing the *right* key and the *wrong* keys which is explained in later sections.

3.3 Truncated Differential Analysis

The classical differential analysis typically follows a differential trail and compute probabilities for known expected differences. In 2012, Albrecht and Leander explained an all-in-one approach to differential analysis [1]. They find probability distribution of the output differences (for the whole state) from one (or more) input difference. In a similar work, multiple differential cryptanalysis using the LLR and χ^2 statistical tests are discussed in [18]. However in [1, 18] the differential distribution is found for the whole state, which makes the attack possible only on a cipher with a *small* block size.

We follow almost the same idea, however we find the differential distribution of every nibble at the state. The way we find the truncated differential distribution in the Markov model, makes our attack possible on the ciphers with larger states than [1, 18].

The analysis is structured in to the following three phases (Figure 3.3 depicts the range of each phase):

1. *Standard Differential (SD) phase* starts from state S_0 with a known input difference $\boldsymbol{\alpha}$, and follows a standard differential trail through SD-rounds up to state S_1 with specific output difference $\boldsymbol{\beta}$.

2. *Truncated Differential Distribution (TDD) phase* calculates the truncated differential distribution from input $\boldsymbol{\beta}$ through TDD-rounds to state S_2 with output $\boldsymbol{\Gamma}$. The output $\boldsymbol{\Gamma}$ here is not a specific difference but a probability distribution over all possible differences.

3. *Partial-Key Recovery (PKR) phase* involves partial decryption of the ciphertext to determine S_2 from the observed output state S_3. The difference in state S_2 is measured and compared against the expected distribution $\boldsymbol{\Gamma}$.

Figure 3.3: The attack model

3.3.1 Standard Differential Phase

The Standard Differential (SD) phase involves finding a high probability differential characteristic through some number of rounds. The XOR-difference between two states $\boldsymbol{x}$ and $\boldsymbol{x}'$ is denoted by $\boldsymbol{\alpha} = (\alpha_{15} \dots \alpha_1 \alpha_0) = (x_{15} \oplus x'_{15}, \dots, x_1 \oplus x'_1, x_0 \oplus x'_0)$. Note that α_i represents exact difference of 4-bits, hence $\alpha_i \in \{0, \dots, 15\}$. The differential trail maps a specific input difference $\boldsymbol{\alpha}$ to a specific output difference $\boldsymbol{\beta}$ with probability denoted $\mathbf{P_{SD}}(\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta})$.

For example, let the input difference be $\boldsymbol{\alpha} = (10000000\ \ 00002000)$. A possible output difference, after one round, is $\boldsymbol{\beta} = (00000000\ \ 10000000)$. The probability of this differential is 2^{-2}.

$$\mathbf{SD} : (10000000\ \ 00002000) \rightarrow (00000000\ \ 10000000) \tag{3.3}$$

The probability is computed under the assumption that the input values of S-box s_7 are not known. If the inputs to the S-box are known, we can detect (with probability 1) whether the differential trail is followed. This requires knowledge of nibble 7 of $SK[0]$. Conversely, given the values of the state, we can find solutions to the sub-key $SK[0]_7$ such that the differential trail is followed.

3.3.2 Truncated Differential Distribution Phase

In the classical differential analysis, we determine the probability that the round function outputs a *specific* difference given a *specific* input difference. In this phase, we model the difference distribution of *all* possible output differences for every nibble based on a chosen distribution of input differences. This generalisation is the fundamental idea behind truncated differential analysis [51] and all-in-one differential analysis [1].

For nibble i, we denote $\boldsymbol{A_i}$ as the probability distribution over all differential values of α_i. It follows that $\boldsymbol{A} = (\boldsymbol{A_{15}} \dots \boldsymbol{A_1 A_0})$ can be represented by a two dimensional matrix with probability distribution of differences for all nibbles.

For example, the following difference distribution matrix states that nibble 0 has difference 15 (with probability 1) and all other nibbles have zero difference (with probability 1).

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A_{15}} & \dots & \boldsymbol{A_1} & \boldsymbol{A_0} \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

By representing the round update function as a differential transformation matrix $\boldsymbol{M}$, we can determine the output differential distribution $\boldsymbol{B}$ from the

input distribution $\boldsymbol{A}$. That is, $\boldsymbol{B} = \boldsymbol{M} \times \boldsymbol{A}$. Assuming a Markov model, we can determine the output differential distribution after n rounds using $\boldsymbol{B} = \boldsymbol{M}^n \times \boldsymbol{A}$.

A Markov cipher is an iterated cipher when the probability distribution of round r only depends on the probability distribution of round $r-1$. Moreover, all round keys are independent and uniformly random. It has been proven in [54] (Theorem 2) that a Feistel cipher with independent round sub-keys is Markov. Although round sub-keys are not independent in LBlock, our experiments show that Markov model gives an accurate estimate of probability distribution when the right-hand half and left-hand half differences are independent. For a limited number of rounds, the dependencies do not cause an issue and this method serves as a reasonable model. We calculated differential distributions for at most 10 rounds of LBlock and all distributions have been empirically verified.

Computing Truncated Differential Distribution

The round function consists of two components that affect the probability distribution, i.e. S-box transformation and XOR addition. Proposition 3.1, describes probability of differences for each nibble after an S-box transformation, and Proposition 3.2 shows how XOR addition affects the difference probability distribution.

Proposition 3.1. *For an S-box $s_n : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ and input difference probability distribution $\boldsymbol{x} = (x^0 x^1 \dots x^{15})$, where x^i is the probability of difference i for nibble n, the output difference probability y^i after S-box transformation s_n is calculated as*

$$y^i = \sum_{j=0}^{15} x^j \cdot \mathbf{P}(s_n(j) = i) \tag{3.4}$$

Proof. Assume the difference J occurs with probability x^J and 4-bit S-box s_n transfers difference J to difference I with probability $\mathbf{P}(s_n(J) = I)$. Hence, difference I happens from input difference J with probability $x^J \cdot \mathbf{P}(s_n(J) = I)$. However, difference I might occur from s-box transformation of the other 15 input differences; therefore output difference I happens with probability y^I as $y^I = \sum_{j=0}^{15} x^j \cdot \mathbf{P}(s_n(j) = I)$. The same way is used to calculate probability y^i for every output difference $0 \leq i \leq 15$. □

Proposition 3.2. *For two input difference probabilities* $\boldsymbol{x} = (x^0 x^1 \dots x^{15})$ *and* $\boldsymbol{y} = (y^0 y^1 \dots y^{15})$*, the output XOR-difference probability* z^i *is*

$$z^i = \sum_{j=0}^{15} x^j \cdot y^{i \oplus j} \tag{3.5}$$

Proof. Assume nibble Z is the XOR-addition of nibbles X and Y. Difference J at nibble X happens with probability x^J; while, in nibble Y, difference $K = I \oplus J$ happens with probability y^K. By XORing differences J and K, nibble Z has difference I with probability $z^I = x^J \cdot y^{I \oplus J}$. However, difference I might be the result of XORing other 15 differences $0 \leq j \leq 15$ of nibble X with difference $k = I \oplus j$ of nibble Y. Thus, difference I happens with probability $z^I = \sum_{j=0}^{15} x^j \cdot y^{I \oplus j}$. For every difference $0 \leq i \leq 15$ of nibble Z probability z^i is calculated with the same way. □

These propositions allow us to construct the differential transformation matrix for the round function; and, given an input distribution, obtain the output truncated differential distribution after a number of rounds. Thus, the TDD phase maps a difference vector $\boldsymbol{\beta}$ to a distribution matrix $\boldsymbol{\Gamma}$. We denote the probability distribution matrix $\mathbf{P_{TDD}}(\boldsymbol{\beta} \rightarrow \boldsymbol{\Gamma})$. For example, let the input difference vector be $\boldsymbol{\beta} =$ (00000000 10000000):

$$\mathbf{TDD} : (00000000\ \ 10000000) \rightarrow (\boldsymbol{\Gamma}_{15}\boldsymbol{\Gamma}_{14}\boldsymbol{\Gamma}_{13}\boldsymbol{\Gamma}_{12}\boldsymbol{\Gamma}_{11}\boldsymbol{\Gamma}_{10}\boldsymbol{\Gamma}_{9}\boldsymbol{\Gamma}_{8}\ \ \boldsymbol{\Gamma}_{7}\boldsymbol{\Gamma}_{6}\boldsymbol{\Gamma}_{5}\boldsymbol{\Gamma}_{4}\boldsymbol{\Gamma}_{3}\boldsymbol{\Gamma}_{2}\boldsymbol{\Gamma}_{1}\boldsymbol{\Gamma}_{0})$$

Table 3.2 lists the output truncated differential distribution $\mathbf{P_{TDD}}(\boldsymbol{\beta} \rightarrow \boldsymbol{\Gamma})$ for the right-hand half nibbles after 8 rounds of LBlock, calculated using Propositions 3.1 and 3.2.

The analysis is more effective if a differential distribution profile is chosen in a way that is easiest to distinguish. More specifically, a distribution that is significantly different from uniformly random. As described in Section 3.2, KL-divergence is the most accurate way to measure the distance between two distributions [6]. The last row in Table 3.2 lists the KL-divergence between calculated probability distribution and uniform distribution for every nibble. Here, U denotes the uniform probability distribution with equal probability $\mathbf{P_U} = 1/16$. Note, from Table 3.2, there are impossible differentials in nibbles 0, 1, 4, 5 and

Table 3.2: Example truncated differential distribution after 8 rounds.

Diff\Nibble	Γ_7	Γ_6	Γ_5	Γ_4	Γ_3	Γ_2	Γ_1	Γ_0
0	0.0610	0.0654	0.0000	0.0000	0.0667	0.0667	0.0000	0.0000
1	0.0000	0.0592	0.0312	0.0693	0.0625	0.0625	0.0625	0.0645
2	0.0649	0.0620	0.1562	0.0732	0.0626	0.0624	0.0312	0.0635
3	0.0649	0.0619	0.0312	0.0684	0.0623	0.0626	0.0938	0.0649
4	0.0610	0.0608	0.0469	0.0698	0.0620	0.0625	0.0625	0.0654
5	0.0732	0.0646	0.0469	0.0610	0.0626	0.0625	0.0625	0.0664
6	0.0703	0.0657	0.0781	0.0649	0.0622	0.0624	0.1250	0.0654
7	0.0684	0.0604	0.1094	0.0698	0.0625	0.0625	0.0625	0.0688
8	0.0703	0.0588	0.0625	0.0635	0.0617	0.0646	0.0625	0.0649
9	0.0679	0.0663	0.0625	0.0649	0.0618	0.0583	0.0625	0.0757
A	0.0659	0.0627	0.0469	0.0635	0.0623	0.0604	0.0312	0.0659
B	0.0649	0.0626	0.0469	0.0728	0.0619	0.0626	0.0312	0.0684
C	0.0615	0.0615	0.0781	0.0659	0.0621	0.0646	0.0625	0.0649
D	0.0679	0.0634	0.1094	0.0654	0.0619	0.0583	0.0625	0.0728
E	0.0693	0.0591	0.0625	0.0620	0.0626	0.0645	0.1250	0.0630
F	0.0684	0.0656	0.0312	0.0654	0.0623	0.0626	0.0625	0.0654
$D(P\|\|U)$	**6.59e-2**	**7.37e-4**	**1.81e-1**	**6.59e-2**	**1.55e-4**	**5.6e-4**	**1.46e-1**	**6.57e-2**

7. This is due to the short number of rounds used in the sample and does not generally occur in longer trails.

3.3.3 Partial Key Recovery Phase

Similar to the classical differential attack, additional rounds are added to the end of the truncated differential distinguisher. In this analysis, the method for distinguishing is based on the distance between a differential distribution P and the uniform distribution U. From the truncated differential distribution table, we choose one (or more) nibbles with significantly large KL-divergence. This nibble we term the *target* nibble and set P equal to the probability distribution for this nibble. By guessing a subset of the round keys and decrypting ciphertext pairs through the final rounds, we observe the target nibble differential distribution. For LBlock, it is not required that the entire sub-key be known to determine nibbles from previous rounds. For example, we choose nibble 3 (of the right-hand

half) as the target nibble. Table 3.3 lists the nibbles required to decrypt 3 rounds and determine nibble 3. The X signifies nibbles that must be calculated in order to decrypt back to the target nibble. The key bits are determined relative to the key register at round $n-3$.

Table 3.3: Nibbles required to be known for decrypting 3 rounds.

Round	Left half nibbles	Right half nibbles	sub-key nibbles	Key bits
n-3	- - - - - - - -	- - - - X - - -	X - - - - - - -	79-78-77-76
n-2	- - X - - - - -	X - - - - - - -	- - - - X - - -	34-33-32-31
n-1	- - - - - - - X -	- - X - X - - -	X X - - - - - -	21-20-19-18 17-16-15-14
n	X - X - - - - -	X X - - - - X -	- - - - - - - -	

For every partial key guess, we decrypt N ciphertext pairs and count the frequency of each difference in the target nibble. The difference frequency is stored in an array of 16 counters $\boldsymbol{c} = (c_0 c_1 \dots c_{15})$. The corresponding probability distribution $\hat{P}$ for this sample is $\hat{P} = 1/N \cdot \boldsymbol{c}$ which allows us to calculate the LLR for each key guess. The LLR is used to determine if the observed data most likely belongs to distribution P or U. If P is chosen in favour of U, the guessed key is considered a potential solution for the real key. Otherwise, it is discarded.

3.3.4 Combining The Three Phases

We merge the standard differential trail of SD with the truncated differential distribution of TDD to achieve a differential profile over an extended number of rounds. Then PKR rounds are added to the end of the trail for the key-recovery attack. From the key schedule, there is a strong dependency between the sub-key bits guessed in PKR and the sub-key bits affecting SD. This changes the success probability $\mathbf{P_{SD}}$. Note there are two S-boxes s_8 and s_9 used in the key scheduling. These S-boxes introduce nonlinear relationships between sub-keys, meaning the PKR key bits are not always directly obtained from SD key bits. We select the SD and PKR phases in a way such that there are as many common bits as possible for the key bits used in the PKR and SD phases.

After combining SD and TDD phases, the expected output difference distribution of TDD is updated due to the success probability of each possible SD

differential output. The probability distribution of differences resulting from the input difference α can be computed as follows:

$$\begin{aligned}\mathbf{P_{TDD}}(\boldsymbol{\alpha} \to \boldsymbol{\Gamma}) &= \sum_i \mathbf{P_{SD}}(\boldsymbol{\alpha} \to \boldsymbol{\beta}_i) \cdot \mathbf{P_{TDD}}(\boldsymbol{\beta}_i \to \boldsymbol{\Gamma}) \\ &= \mathbf{P_{SD}}(\boldsymbol{\alpha} \to \boldsymbol{\beta}_j) \cdot \mathbf{P_{TDD}}(\boldsymbol{\beta}_j \to \boldsymbol{\Gamma}) \\ &+ \sum_{i \neq j} \mathbf{P_{SD}}(\boldsymbol{\alpha} \to \boldsymbol{\beta}_i) \cdot \mathbf{P_{TDD}}(\boldsymbol{\beta}_i \to \boldsymbol{\Gamma}),\end{aligned} \tag{3.6}$$

where $\boldsymbol{\beta_i}$ are all possible output difference vectors of the SD phase. In Equation (3.6), $\boldsymbol{\beta}_j$ is the input difference for the truncated differential distribution TDD that has the most distinguishable profile (highest KL-divergence). Usually, $\boldsymbol{\beta}_j$ is the difference with the lowest hamming weight. Also, in practice, all other $\boldsymbol{\beta}_i$ lead to probability distributions that are much closer to uniform (in comparison to $\boldsymbol{\beta}_j$). That is,

$$\sum_{i \neq j} \mathbf{P_{SD}}(\boldsymbol{\alpha} \to \boldsymbol{\beta}_i) \cdot \mathbf{P_{TDD}}(\boldsymbol{\beta}_i \to \boldsymbol{\Gamma}) \approx (1 - \mathbf{P_{SD}}(\boldsymbol{\alpha} \to \boldsymbol{\beta}_j)) \cdot \mathbf{P_U} \tag{3.7}$$

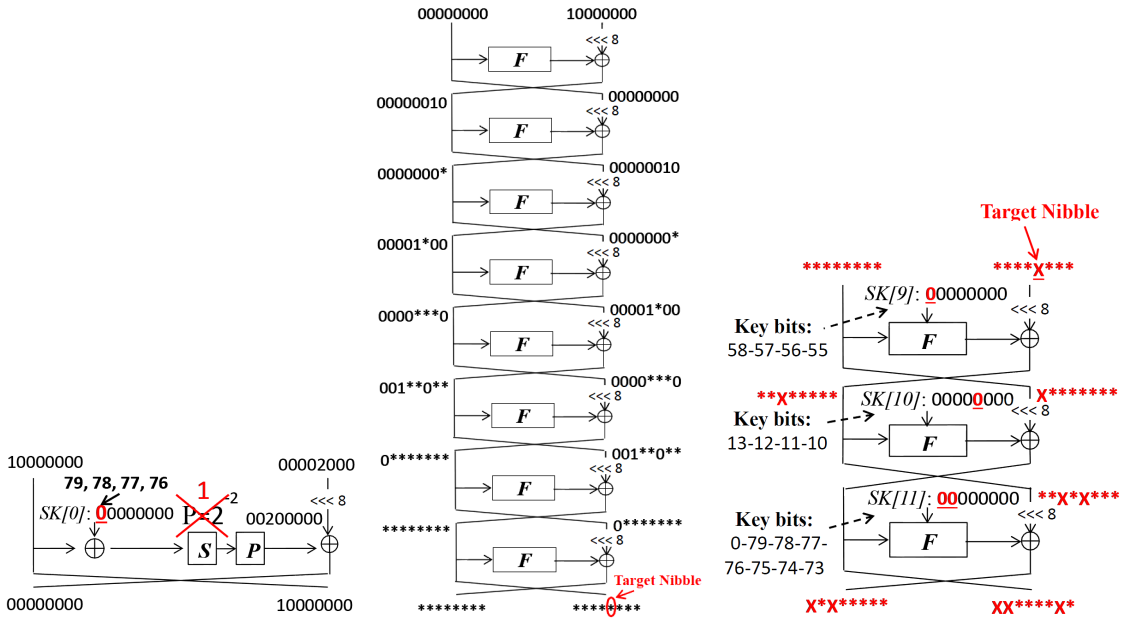
From (3.6) and (3.7), the output probability distribution is approximated by

$$\mathbf{P_{TDD}}(\boldsymbol{\alpha} \to \boldsymbol{\Gamma}) \approx \mathbf{P_{SD}}(\boldsymbol{\alpha} \to \boldsymbol{\beta}_j) \cdot \mathbf{P_{TDD}}(\boldsymbol{\beta}_j \to \boldsymbol{\Gamma}) + (1 - \mathbf{P_{SD}}(\boldsymbol{\alpha} \to \boldsymbol{\beta}_j)) \cdot \mathbf{P_U} \tag{3.8}$$

3.3.5 12-Round Example Attack

This section gives details about how the analysis is applied to a 12-round version of LBlock. We construct a 9-round differential distinguisher by combining the 1-round $\mathbf{SD}(\boldsymbol{\alpha} \to \boldsymbol{\beta})$ (from Equation(3.3)) with 8-round $\mathbf{TDD}(\boldsymbol{\beta} \to \boldsymbol{\Gamma})$ (from Section 3.3.2). Three additional rounds are added for the PKR phase (described in Section 3.3.3). The entire attack structure is depicted in Figure 3.4.

To cover the general application of the analysis, we choose nibble 3 as the target nibble for the PKR phase, which does not benefit from the impossible dif-



(a) SD phase **(b)** TDD phase **(c)** PKR phase

Figure 3.4: 3 phases of the 12-round example attack

ferential. The sub-keys required to decrypt the ciphertext in the PKR phase (i.e. the underlined sub-key nibbles in Figure 3.4c) include $SK[11]_7, SK[11]_6, SK[10]_3$ and $SK[9]_7$, a total of 16 unique bits. The sub-key used in SD phase is $SK[0]_7$. From the key schedule we get

$$SK[0]_7 = ((s_9^{-1}(SK[11]_7)\ \&\ \texttt{0x7}) \ll 1)\ |\ (s_8^{-1}(SK[11]_6)\ \&\ \texttt{0x1}).$$

Table 3.4 depicts the input key bits to the key scheduling S-boxes s_9 and s_8 for 12 rounds. The key bits are numbered relative to the round 0 master key register. It is clear in the table, that key bits in $SK[0]_7$, are guessed in the last round. Although these bits are in the two most significant nibbles of the key register and should pass through the inverse S-boxes, their value is known and so the input bits to the S-boxes. Hence, for a given guess in PKR phase, we determine the sub-key used in the SD phase with no extra effort.

For a chosen input plaintext pair (with difference $\boldsymbol{\alpha}$), we say it is a *right-pair* if it follows the differential SD. Otherwise, the pair is termed a *wrong-pair*. Note that the attacker does not have access to the internal differential states, he

Table 3.4: Input bits to the key scheduling S-boxes.

Round	S-box s_9 input bits	S-box s_8 input bits
0	**79-78-77-76**	75-74-73-72
1	50-49-48-47	46-45-44-43
2	21-20-19-18	17-16-15-14
3	72-71-70-69	68-67-66-65
4	43-42-41-40	39-38-37-36
5	14-13-12-11	10- 9- 8- 7
6	65-64-63-62	61-60-59-58
7	36-35-34-33	32-31-30-29
8	7- 6- 5- 4	3- 2- 1- 0
9	58-57-56-55	54-53-52-51
10	29-28-27-26	25-24-23-22
11	**0-79-78-77**	**76-75-74-73**

only sees the ciphertext pair. For random input pairs, $\mathbf{P_{SD}}(\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}) = 2^{-2}$, and we expect 1/4 right-pairs on average. Henceforth, we denote the total number of plaintext pairs N_p and the number of right-pairs N. For every guess of key bits in PKR, we determine $SK[0]_7$ and distinguish right-pairs from wrong-pairs (with respect to the key guess). By disregarding wrong-pairs we can increase the probability of the SD phase such that $\mathbf{P_{SD}}(\boldsymbol{\alpha} \rightarrow \boldsymbol{\beta}) = 1$. Therefore, from Equation (3.8), $\mathbf{P_{TDD}}(\boldsymbol{\alpha} \rightarrow \boldsymbol{\Gamma}) = \mathbf{P_{TDD}}(\boldsymbol{\beta} \rightarrow \boldsymbol{\Gamma})$.

When $SK[0]_7$ is incorrect (due to an incorrect guess in PKR), we mistake a wrong-pair for a right-pair. This false-positive results in the addition of noise to the observed probability distribution. The noise is assumed to be uniformly random, a similar assumption to the Wrong Key Randomization Hypothesis [54] (explained later). However, this false-positive only occurs for incorrect guesses and does not affect the correct guess distribution.

3.4 Complexity Analysis

For every key guessed in the PKR phase, we calculate the LLR between the observed truncated differential distribution and the expected one. If the LLR is above some threshold (Θ), we consider the guessed key a candidate for the right

key. The resulting list of candidate keys are checked for correctness. The attack is successful *if* the right key is among the list of candidate keys, we call this the *attack success rate*. In [1], the "gain" of the attack is the fraction of wrong keys ranked above the expected rank of the right key. We extend this concept and determine the expected number of candidate keys and the effort required to find the right key among them.

For every input-output difference, the counter for the guessed keys after decrypting N pairs of ciphertexts, follows a binomial distribution [27]. So the distribution of the the guessed key counters for all the possible output differences is a multinomial distribution [1]. Now, assume R is a random variable for the LLR of the right candidate. the expected count for the right candidate is defined by $E(R)$ in Equation (3.9). Likewise, random variable W is defined for the wrong candidates. The value $E(W)$ gives the expected count of the wrong candidate, defined in Equation (3.10).

$$E(R) = N \sum_i p_i \, ln(\frac{p_i}{q_i}) \tag{3.9}$$

$$E(W) = N \sum_i q_i \, ln(\frac{p_i}{q_i}) \tag{3.10}$$

here N is the number of right-pairs, p_i is the probability of getting the difference i under the right key (which is found in the TDD phase), and q_i is the probability of getting the difference by a wrong key. According to the *Wrong Key Randomization Hypothesis* [54], difference probabilities after decryption by a wrong key candidate are distributed as for a random permutation. Our experiments on LBlock confirm the hypothesis for two or more rounds of decryption.

LLR distribution of the right key is approximated by a normal distribution with a mean of $E(R)$ and variance of $Var(R)$ defined in Equation (3.11) [1]. Likewise, the average distribution of the wrong keys, is approximated by another normal distribution with a mean of $E(W)$ and variance of $Var(W)$, given in Equation (3.12). Figure 3.5 shows the normal distributions of both right key and wrong keys LLRs.

$$Var(R) = N\left(\left(\sum_i p_i\left(ln(\frac{p_i}{q_i})\right)^2\right) - \left(\sum_i p_i ln(\frac{p_i}{q_i})\right)^2\right) \tag{3.11}$$

$$Var(W) = N\left(\left(\sum_i q_i\left(ln(\frac{p_i}{q_i})\right)^2\right) - \left(\sum_i q_i ln(\frac{p_i}{q_i})\right)^2\right) \tag{3.12}$$

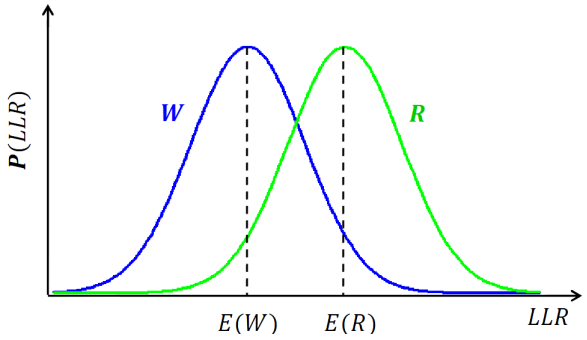


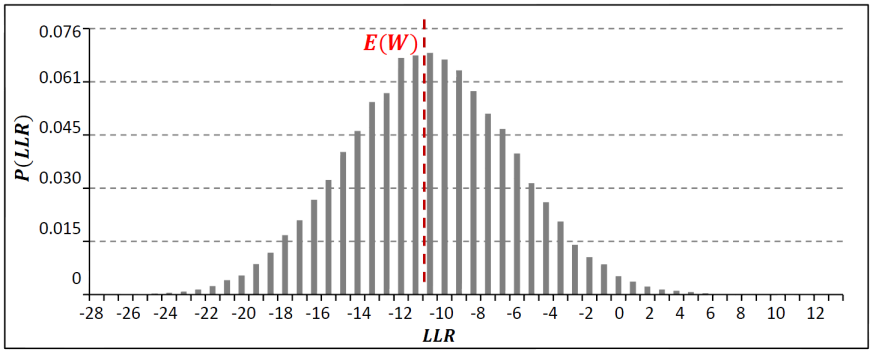
Figure 3.5: LLR distributions of R and W

To verify the theoretical findings by experiments, we implemented the analysis on the 12-round example of Section 3.3.5. We ran the analysis 1000 times with $N = 2^{16}$ right-pairs each, and found the LLR distribution for random variables R and W. Note in this example we guess 16 key bits in the PKR phase, therefore there are 2^{16} candidate keys. Fig. 3.6b shows the LLR distribution for the right key from the experiments. Likewise, Fig. 3.6a shows the average LLR distribution of all the wrong keys. The theoretical values describing these distributions are, $E(R) = 10.2242$, $E(W) = -10.0356$, $Var(R) = 20.8225$, and $Var(W) = 19.7064$.

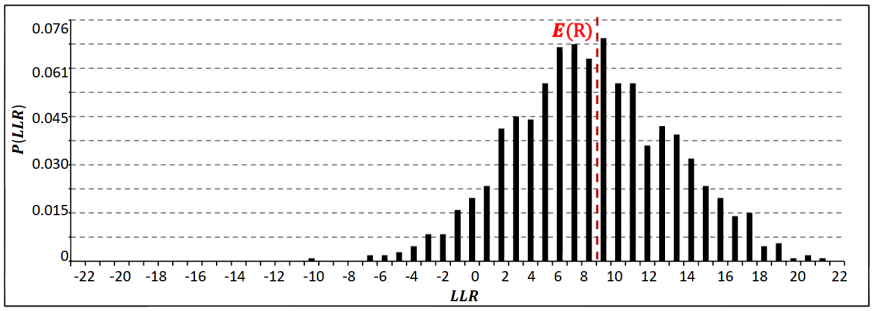
Assume random variable X follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$, where μ and σ^2 are the mean and variance, respectively. According to the cumulative distribution function (CDF), the probability of the random variable X falling into the interval $[x, \infty)$ is (erf is the error function of the distribution):

$$\mathbf{P}(X \geq x) = \frac{1}{2}\left(1 - \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right) \tag{3.13}$$

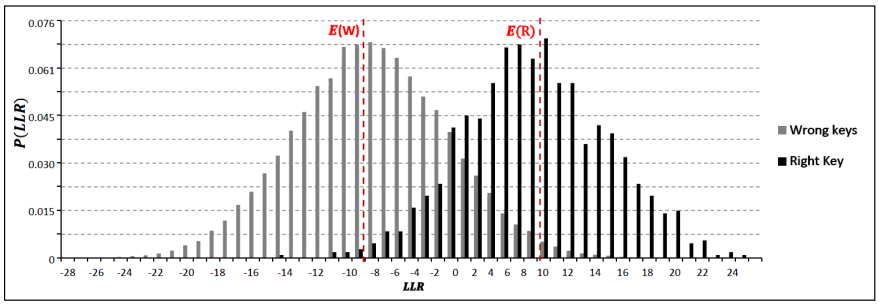
If Θ represents a threshold for the LLR, $\mathbf{P}(R \geq \Theta)$ gives the probability that the right key LLR is greater than the threshold. Likewise, $\mathbf{P}(W \geq \Theta)$ gives the



(a) Average LLR distribution of the wrong keys



(b) LLR distribution of the right key



(c) Combined diagrams

Figure 3.6: Empirical diagrams of the LLR distributions of the 12-round example

probability of a wrong key LLR greater than the threshold Θ. Both probabilities are calculated from Equation (3.13). Since $E(R)$ is the mean for the normal distribution of the expected right key, the right key LLR is higher than $E(R)$ with probability $\frac{1}{2}$. While $\mathbf{P}(W \geq E(R))$ gives the probability of a wrong key being ranked higher than the expected right key. If there are N_K key candidates in the test, N_{wk} denotes the wrong keys ranked higher than the threshold. The expected value of N_{wk} is

$$N_{wk} = N_K \cdot \mathbf{P}(W \geq \Theta) \tag{3.14}$$

The attack success rate for finding the right key is related to the threshold Θ and N the number of right-pairs (accounting for the SD phase) used in the attack. By adjusting Θ and N, the attacker is able to find a higher success rate or a lower N_{wk}.

In the 12-round example attack, we choose $N_p = 2^{18}$ chosen plaintext/ciphertext pairs and expect to get $N = 2^{16}$ right-pairs from the SD phase. We ran the experiments 100 times for each chosen threshold. Table 3.5 shows the results for different success rates by selecting various LLR thresholds. It is clear in Table 3.5 that the experiments confirm the theory.

Table 3.5: 12-round LBlock results for $N = 2^{16}$ right-pairs

Θ	$\mathbf{P}(R \geq \Theta)$	$\mathbf{P}(W \geq \Theta)$	N_{wk}	Empirical $\mathbf{P}(R \geq \Theta)$	Average empirical N_{wk}
2.6189	0.95	0.0021	143	0.94	154.07
5.6610	0.84	0.0002	14	0.87	15.16
7.1821	0.74	5.25e-05	4	0.73	3.68
8.7032	0.63	1.21e-05	0.79	0.61	0.92
10.2242	0.5	2.51e-06	0.16	0.45	0.19

Time complexity of the attacks can be computed in three steps. First step is complexity of finding the right pairs from known key bits in the SD phase. Assume b_c is the number of common bits between SD and PKR phases, and r_{SD} is the number of SD rounds where the common key bits are used to pass the S-boxes. The time complexity of this step respecting one full r-round encryption is,

$$C_{SD} = N_p \, 2^{b_c} \frac{2\, r_{SD}}{r} \tag{3.15}$$

The second step is guessing the remaining PKR key bits (b_P is the number of PKR-key bits), decrypting the tight pairs through r_{PKR} final rounds, and calculating the LLR distribution for every candidate key. The second step time complexity (respecting r rounds encryption) is

$$C_{PKR} = N \, 2^{b_p} \frac{2\, r_{PKR}}{r} \tag{3.16}$$

The complexity of the distinguisher is the sum of the above complexities for the first two steps. In the final step, after the partial key-recovery, each candidate key from the previous step should be checked for correctness to do the full key-recovery. Also the remaining key bits must be found. One naive method is to guess the remaining unknown key bits by exhaustive search. The time complexity of this step is

$$C_{KR} = (N_{wk} + 1)\, 2^{80-b_P} \tag{3.17}$$

The total time complexity of the key-recovery attack respecting r rounds encryption is the sum of the complexity of the above three steps (i.e. Equations (3.15),(3.16) and (3.17))

$$C = C_{SD} + C_{PKR} + C_{KR} \tag{3.18}$$

In the 12-round attack, by choosing $N_p = 2^{18}$ plaintext pairs (results in 2^{16} right-pairs) the distinguisher complexity is

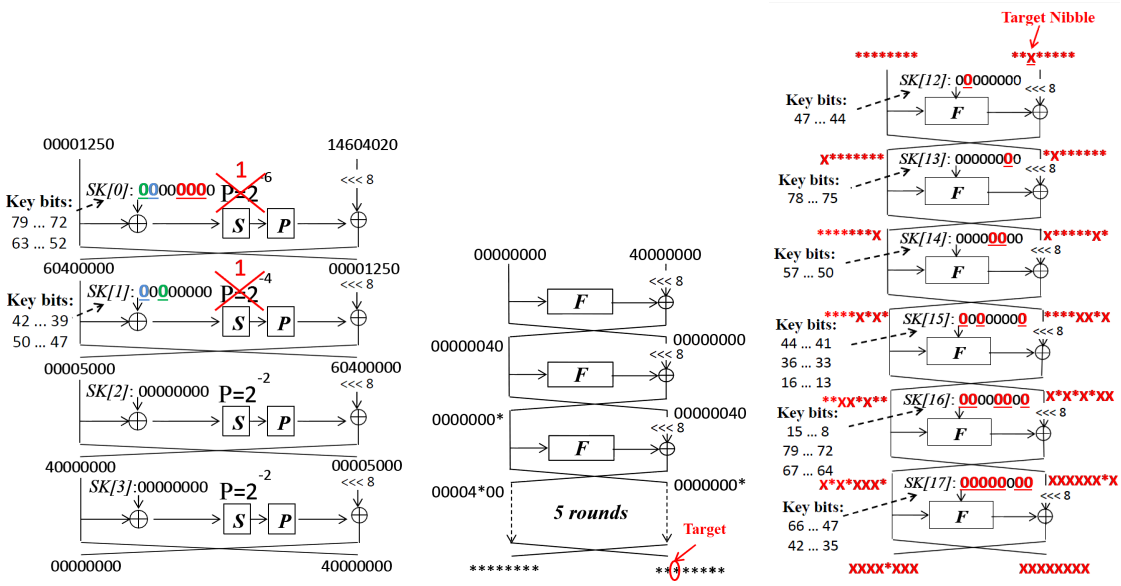
$$C_{SD} + C_{PKR} = 2^{18} \times 2^4 \times \frac{2}{12} + 2^{16} \times 2^{16} \times \frac{6}{12} \backsimeq 2^{31}$$

which is about 2^{31} 12-round encryptions. However the whole key recovery attack time complexity is $C = 2^{31} + 2^{64} \backsimeq 2^{64}$ 12-round encryptions. Here, the exhaustive search of the remaining key bits dominates the complexity. There are more efficient methods for recovering the remaining bits. In cases where the initial phase is the dominant task, the exhaustive search may be used to guess the remaining key bits, as it does not significantly increase the total complexity.

3.5 Key-Recovery Attacks on LBlock

3.5.1 Single-Key Attack on 18 Rounds

Fig. 3.7 describes the truncated differential attack on 18-round LBlock. We divide the 18 rounds into 3 parts to apply the attack. The SD phase takes the first 4 rounds, the TDD phase is the next 8 rounds, and the PKR phase includes the 6 final rounds.



(a) SD phase **(b)** TDD phase **(c)** PKR phase

Figure 3.7: Truncated differential attack on 18-round LBlock

The input state of the 4-round SD phase includes 3 nibbles with non-zero differences in the left-hand half and 5 nibbles with non-zero differences in the right-hand half as shown in Fig. 3.7a. Through the 4-round standard differential almost all the differences are cancelled. So the output state has difference zero in all the nibbles except nibble 7 of the right-hand half. The TDD phase is very similar to that explained in the 12-round attack. It starts with a low weight state (with only difference 4 at nibble 7). The truncated differential distribution is calculated through 8 rounds to get the right-hand half distribution at the output state. The highest KL-divergence belongs to nibble 5 of the right-hand half, which is $D(P||Q) = 2.184e - 01$. Therefore, nibble 5 is chosen as the target nibble for the 6-round PKR phase. To find the LLR distribution for the target nibble, the attacker needs to guess 52 key-bits in the PKR phase.

Observing the SD phase, if the attacker knows the values of 3 sub-key nibbles $SK[0]_1$, $SK[0]_2$ and $SK[0]_3$, he is able to find the output of the 3 active S-boxes in the first round with no extra effort. Likewise, by knowing the values of sub-key nibbles $SK[0]_6$, $SK[0]_7$, $SK[1]_5$ and $SK[1]_7$, he finds the output of 2 active S-boxes in the second round. Table 3.6 shows the required sub-keys for the attack,

Table 3.6: Required round Sub-keys for the 18-round attack .

r#	Nib 7(s_9)	Nib 6(s_8)	Nib 5	Nib 4	Nib 3	Nib 2	Nib 1	Nib 0
0	79 78 77 76	75 74 73 72	71 70 69 68	67 66 65 64	63 62 61 60	59 58 57 56	55 54 53 52	51 50 49 48
1	50 49 48 47	46 45 44 43	42 41 40 39	38 37 36 35	34 33 32 31	30 29 28 27	26 25 24 23	22 21 20 19
2	21 20 19 18	17 16 15 14	13 12 11 10	9 8 7 6	5 4 3 2	1 0 79 78	77 76 75 74	73 72 71 70
3	72 71 70 69	68 67 66 65	64 63 62 61	60 59 58 57	56 55 54 53	52 51 50 49	48 47 46 45	44 43 42 41
4	43 42 41 40	39 38 37 36	35 34 33 32	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12
5	14 13 12 11	10 9 8 7	6 5 4 3	2 1 0 79	78 77 76 75	74 73 72 71	70 69 68 67	66 65 64 63
6	65 64 63 62	61 60 59 58	57 56 55 54	53 52 51 50	49 48 47 46	45 44 43 42	41 40 39 38	37 36 35 34
7	36 35 34 33	32 31 30 29	28 27 26 25	24 23 22 21	20 19 18 17	16 15 14 13	12 11 10 9	8 7 6 5
8	7 6 5 4	3 2 1 0	79 78 77 76	75 74 73 72	71 70 69 68	67 66 65 64	63 62 61 60	59 58 57 56
9	58 57 56 55	54 53 52 51	50 49 48 47	46 45 44 43	42 41 40 39	38 37 36 35	34 33 32 31	30 29 28 27
10	29 28 27 26	25 24 23 22	21 20 19 18	17 16 15 14	13 12 11 10	9 8 7 6	5 4 3 2	1 0 79 78
11	0 79 78 77	76 75 74 73	72 71 70 69	68 67 66 65	64 63 62 61	60 59 58 57	56 55 54 53	52 51 50 49
12	51 50 49 48	47 46 45 44	43 42 41 40	39 38 37 36	35 34 33 32	31 30 29 28	27 26 25 24	23 22 21 20
13	22 21 20 19	18 17 16 15	14 13 12 11	10 9 8 7	6 5 4 3	2 1 0 79	78 77 76 75	74 73 72 71
14	73 72 71 70	69 68 67 66	65 64 63 62	61 60 59 58	57 56 55 54	53 52 51 50	49 48 47 46	45 44 43 42
15	44 43 42 41	40 39 38 37	36 35 34 33	32 31 30 29	28 27 26 25	24 23 22 21	20 19 18 17	16 15 14 13
16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0	79 78 77 76	75 74 73 72	71 70 69 68	67 66 65 64
17	66 65 64 63	62 61 60 59	58 57 56 55	54 53 52 51	50 49 48 47	46 45 44 43	42 41 40 39	38 37 36 35

also it depicts how the key scheduling S-boxes affect the common bits between SD and PKR guessed keys. Key bits are numbered respecting the first round key register. Overall, the attacker needs to know the values of 28 key bits in the PKR phase. These bits are guessed in PKR phase, however going through the key scheduling process the values of bits 73 and 72 are lost. By re-guessing these key bits and guessing one more (bit 0), all the required 28 bit values are revealed for the SD phase. Therefore, the probability of the SD phase is increased to $\mathbf{P_{SD}} = 2^{-4}$.

As mentioned in Section 3.3.4, difference probability distribution is updated after combining the SD and TDD phases estimated by Equation (3.8). Probability distribution of the target nibble 5, is shown in Table 3.7 before and after combining with the SD phase ($\mathbf{P_{TDD}}$ and $\mathbf{P^*_{TDD}}$, respectively).

Adjusting N in Equations (3.9) and (3.10), the attacker finds $N = 2^{13}$ as the value with the best trade off between success rate and complexity. The statistical characteristic of the right key and the wrong key distributions are as follows: $E(R) = 6.44$, $E(W) = -6.40$, $Var(R) = 12.99$, and $Var(W) = 12.71$.

Table 3.7: Difference probability distribution of the target nibble

Difference	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
$\mathbf{P_{TDD}}$	0.000	0.156	0.031	0.093	0.046	0.046	0.015	0.109
$\mathbf{P^*_{TDD}}$	0.058	0.068	0.060	0.064	0.061	0.061	0.059	0.065
Difference	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
$\mathbf{P_{TDD}}$	0.078	0.109	0.031	0.062	0.093	0.031	0.046	0.046
$\mathbf{P^*_{TDD}}$	0.063	0.065	0.060	0.062	0.064	0.060	0.061	0.061

Table 3.8: Analysis results of 18-round LBlock for 2^{23} plaintext pairs

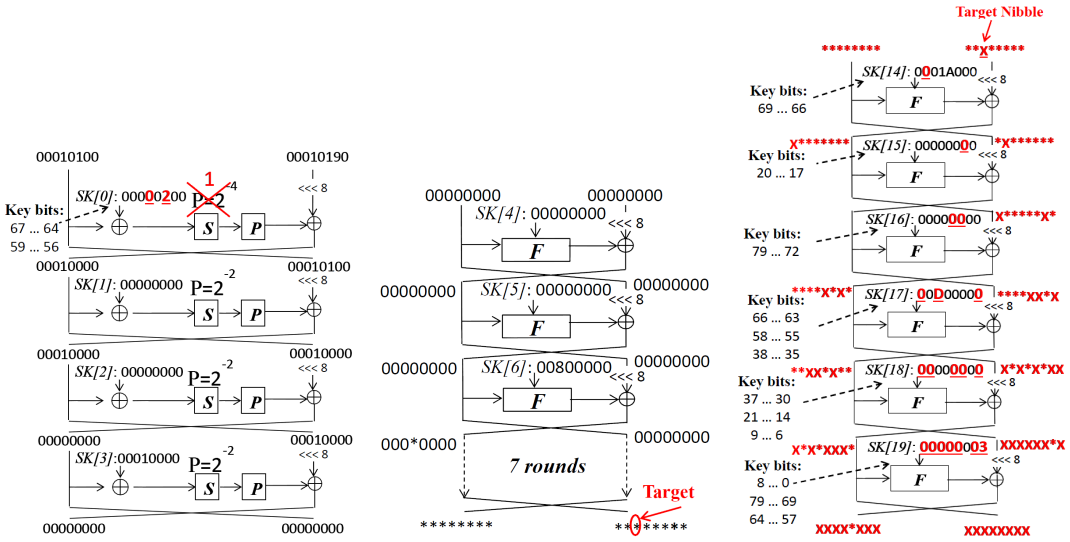
Θ	$\mathbf{P}(R \geq \Theta)$	$\mathbf{P}(W \geq \Theta)$	N_{wk}	Time Complexity
1.043	0.93	0.018	6.62e+14	$2^{74.24}$
2.245	0.87	0.007	2.75e+14	$2^{72.99}$
3.446	0.79	0.002	1.033e+14	$2^{71.63}$
4.647	0.69	0.0009	3.49e+13	$2^{70.21}$
6.449	0.5	0.0001	5.63e+12	$2^{68.38}$

Table 3.8 shows the result on 18-round key-recovery attack with different chosen thresholds. Note, the number of plaintext pairs includes those satisfying the first two rounds of the SD phase (i.e. 2^{10}). Therefore, we need $N_p = 2^{13+10} = 2^{23}$ pairs of plaintext/ciphertext to apply the attack. If the attacker chooses the threshold $\Theta = E(R)$, the probability that he finds the right key is 50% and the attack complexity is $2^{68.38}$.

3.5.2 Related-Key Attacks on 20 and 21 Rounds

The related key truncated differential attack applies to LBlock reduced to 20 and 21 rounds. Considering the key scheduling process, when the key difference goes through the S-boxes s_8 or s_9 the output difference is unknown. However, due to the slow avalanche effect of the key schedule, it takes multiple rounds for key differences to reach these S-boxes. Therefore, it is easy to find the truncated difference probability distribution for all the possible key differentials. During the attack, we test each expected key differential in parallel to determine the correct

key differential path.



(a) 4-round SD phase **(b)** 10-round TDD phase **(c)** 6-round PKR phase

Figure 3.8: Related-key truncated differential attack on 20-round LBlock

The related key attack on 20 rounds consists of a 4-round SD phase, 10-round TDD, and 6-round PKR phase (see Fig. 3.8). The SD phase starts with 5 non-zero differences which are all cancelled through the 4 rounds differential trail, hence finishing with no difference in the output state. The key register at the first round of TDD phase has difference in just one bit (i.e. the 13th least significant bit). The key difference does not affect the round sub-keys for two rounds. The truncated differential distribution is calculated for the 10-round TDD phase. Nibble 5 (of the output right-hand half) has the highest KL-divergence $D(P||Q) = 2.189429e-03$ and is chosen as the target nibble. Finally, 6 final rounds are added as the PKR phase, requiring 52 key bits be guessed to reach the target nibble. From these key bits, two sub-key nibbles $SK[0]_2$ and $SK[0]_4$ are determined for the first round of the SD phase (i.e. 8 common key bits between SD and PKR phases). Consequently, the input values of the active S-boxes are known in the first round and the overall probability of the SD phase increases to $\mathbf{P_{SD}} = 2^{-6}$.

Table 3.9, shows the results for the 20-round related key attack with different success rates. Note that the number of plaintext/ciphertext pairs includes the amount required to follow the SD phase. Considering the LLR threshold equal to

the expected value of the right key ($E(R)$), with 2^{27} chosen plaintexts ($N = 2^{23}$ right-pairs), the complexity of the key recovery attack is $2^{74.33}$.

Table 3.9: Related-key analysis results on reduced-round LBlock

Specification	Θ	$\mathbf{P}(R \geq \Theta)$	$\mathbf{P}(W \geq \Theta)$	N_{wk}	Time Complexity
20 rounds,	1.6798	0.84	0.0183	8.28e+13	$2^{75.24}$
$N_p = 2^{27}$ pairs,	3.7397	0.63	0.0029	1.31e+13	$2^{74.47}$
$E(R) = 4.7696$	4.7696	0.5	0.0010	4.51e+12	$2^{74.33}$
21 rounds,	-0.1320	0.74	0.3355	4.83e+16	$2^{78.61}$
$N_p = 2^{30}$ pairs,	0.2320	0.63	0.2240	3.23e+16	$2^{78.11}$
$E(R) = 0.5962$	0.5962	0.5	0.1373	1.98e+16	$2^{77.56}$

The related-key attack is extended to 21 rounds by adding one more round to the beginning of the SD phase in the above 20-round attack. Fig. 3.9 shows the SD phase in 21-round attack. The other phases are similar to the ones in the 20-round attack. If the attacker guesses 5 more key bits in the PKR phase (a total of 57 bits), he finds the 3 sub-key nibbles ($SK[0]_1$, $SK[0]_2$ and $SK[0]_4$) required to know the values of the active S-boxes in the first SD round. Also, to know the input values of 2 active S-boxes in the second round sub-key nibbles $SK[0]_0$, $SK[0]_5$, $SK[1]_2$ and $SK[1]_4$ must be known (total number of 28 common key bits between SD and PKR phases). By having $N_p = 2^{30}$ chosen plaintexts, there are $N = 2^{20}$ right-pairs for the key-recovery attack. The analysis results of 21 rounds with different success rates are shown in Table 3.9. Overall, the related-key attack on 21-round LBlock is possible with $N_p = 2^{30}$ chosen plaintext/ciphertext pairs and $2^{77.56}$ time complexity, when the attack success rate is 50%.

3.6 Conclusion

In this chapter we presented truncated differential analysis of the lightweight block cipher LBlock by analysing probability distribution of the truncated differences. We used LLR statistical test to distinguish and apply key-recovery attacks. The attack uses a truncated differential distribution that is significantly different from a random permutation. Candidate sub-keys are guessed over several final rounds

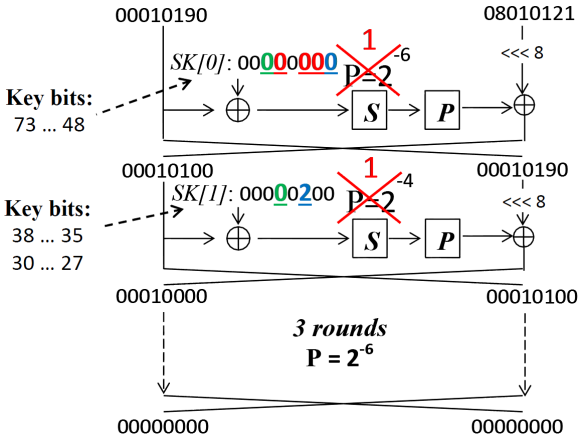


Figure 3.9: The SD phase of the 21-round related-key attack

and the observed differences are measured against the expected distribution. We extend the distinguisher by concatenating additional rounds to the beginning which follow a classical differential characteristic. By exploiting the properties of the key schedule, we greatly increase the probabilities of differentials passing through the beginning rounds. We verify the analysis by implementing an example attack on 12-round LBlock and provide empirical data confirming the theory. Finally, we describe single-key and related-key attacks on LBlock reduced to 18 and 21 rounds, respectively. Finding probability distribution of the truncated differential, our attack can be applied on the ciphers with relatively large block size.

4

Rebound Attack on Feistel Networks and Application to Camellia

Feistel Networks are widely used to construct block ciphers and hash functions. Feistel structures can be analysed by the rebound attack as a truncated differential method. The rebound technique is an efficient way to find known-key distinguishers on Feistel ciphers and collision attacks on the hash functions. Known-key distinguishers, proposed by Knudsen and Rijmen [52], detects non-ideal properties of a random instantiation of a fixed permutation, while the same properties cannot be observed in a random permutation with the same complexity. In a separate work, Sasaki and Yasuda present a known-key distinguisher using the rebound attack on Feistel ciphers with SPN round functions (we call them Feistel-SP networks) [93].

In this chapter, first we introduce the rebound attack and explain the preliminary concepts. Then we improve the rebound attack on Feistel-SP networks, where the inbound phase includes 5 rounds. Using the proposed method, we at-

tack Camellia-128 block cipher and find a distinguisher on the cipher reduced to 11 rounds. Also in the hashing mode, we find 9-round collision and half-collision attacks, as well as 4-sum distinguishers for 7 and 9 rounds. At the end, we verify our results on Camellia-128 experimentally.

4.1 Preliminaries

4.1.1 The Rebound Attack

The rebound attack is a truncated differential method proposed by Mendel *et. al.* [73] for the analysis of hash functions and block ciphers. They introduced the rebound analysis on AES-like designs and used it to launch collision attacks on Grøstl-0 [37] and Whirlpool [7] hash functions. In the rebound attack, the cipher E is split into three sub-ciphers E_{bw}, E_{in} and E_{fw}, where $E = E_{fw} \circ E_{in} \circ E_{bw}$. Then, as shown in Figure 4.1, the attack employs truncated differentials between the sub-ciphers round transformations in an *inside-out* approach via the *inbound* and *outbound* phases.

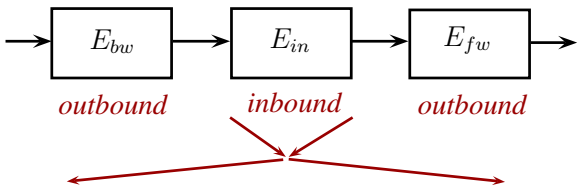


Figure 4.1: The rebound attack schematic

The inbound phase follows the meet-in-the-middle technique and starts with low-weight truncated differences on the states on both sides. The truncated differences are propagated forward and backward to meet each other in the middle. This phase is the most expensive part of the analysis and exploits degrees of freedom to find efficient matches. The solutions of the inbound phase are pairs of values conforming the inbound phase truncated differential trail. The outbound phase is the probabilistic part of the attack and extends the truncated differential of the inbound phase sides in forward and backward directions. Solutions of the inbound phase are used as the starting points for the *outbound* phase.

Assume an AES-like SP cipher with a 16-byte state represented by a 4×4 matrix. The linear permutation of this cipher is a *maximum distance separable (MDS)* matrix. A MDS matrix [98] is a $r \times r$ matrix when its branch number is $r + 1$. So if it multiplies by a state with at least one active cell (e.g. non-zero difference) in the input state, the total number of active cells in the input and output truncated differences is always greater than or equal to $r + 1$. Figure 4.2 depicts the inbound phase where each side starts with 4 active bytes. The inbound phase goal is to find pairs of values that satisfy its truncated differential. We denote the truncated differential path $\mathbf{4} \rightarrow \mathbf{8} \rightarrow \mathbf{4}$. It means 4 active bytes propagate to a full-active state with 8 non-zero differences and then converge to 4 active bytes. The inbound phase works as follows:

1. The difference distribution table (DDT) is calculated for the 8-bit S-box (or S-boxes).

2. For all $(2^8 - 1)^4 \backsimeq 2^{32}$ possible differences of state S_0, all the corresponding differences in state S_1 are computed and stored in table T.

3. A possible difference at state S_3 is chosen and computed backward through the linear permutation to state S_2. Then for each 8-byte difference in table T the DDT is checked to see if the difference of state S_2 can be generated from either of differences in state S_1, if such a differential is found the corresponding values are considered as a solution for the inbound phase.

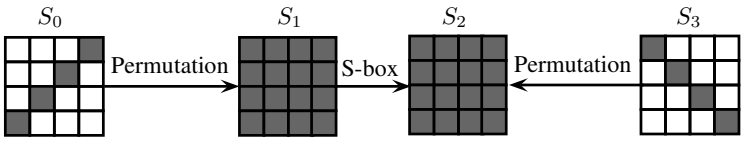


Figure 4.2: The inbound phase for an AES-like cipher

The AES S-box transfers a random input difference to a random output difference with probability of approximately 1/2 and has one solution. Therefore, looking at all the possible differences of state S_0, we find at least one pair of values which satisfy the difference of state S_3. So, we expect to get about 2^{32}

solutions for the inbound phase with 2^{32} pre-computations. Hence, on average every solution is found with complexity of $O(1)$ computation. Besides, there are 2^{32} possible differences for the state S_3. Thus, totally we can generate $2^{32} \cdot 2^{32} = 2^{64}$ solutions (pairs of values) for the inbound phase, with 2^{32} computations and 2^{32} states of memory.

4.1.2 Feistel-SP Networks

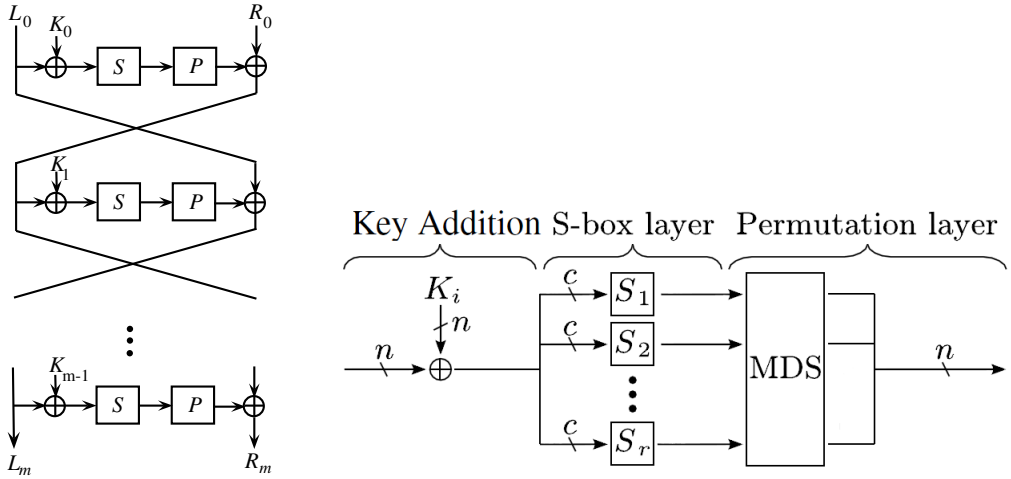
Figure 4.3a shows a m-round Feistel network with a substitution-permutation round function, called the *Feistel-SP network*. The round function consists of key addition, a S-box layer and then a permutation layer (see Figure 4.3b). We use the following notations to describe the SP round function:

N: The block length of the cipher (in bits),

n: The word size in bits, equal to the size of the round function ($n = N/2$).

c: The size of an S-box in bits,

r: The number of S-box sequences, hence $r = n/c$.



(a) Feistel-SP structure **(b)** A SP round function

Figure 4.3: Feistel-SP Network

The components of the SP round function are typically defined as follows:

- *Key Addition*: The round sub-key K_i is added to the round-function input (usually by the XOR addition).

- *S-box layer*: This layer consists of r S-boxes working in parallel. Each input value is substituted with the output value by an S-box transformation. The S-boxes $S_1, S_2, \ldots, S_r$ may differ from each other. For simplicity, we assume that the S-box is designed to resist differential and linear cryptanalysis, such as the AES S-box [26].

- *Permutation layer*: The linear diffusion is introduced to output sequences of the S-boxes. This layer mixes values and multiplies the input value by an $r \times r$ matrix P over $\mathbb{F}_2^c$. We make the assumption that P is a (MDS) matrix, so its branch number is $r + 1$.

Note that the assumptions on S and P are not necessary. For example, Whirlpool [7] adopts a more biased S-box than AES, however Lamberger *et. al.* [56] showed that the rebound attack for Whirlpool can work similarly to AES. Additionally, the Camellia permutation matrix is not MDS, but we show later that our attack can be applied on this cipher with some modifications.

4.2 Related Work

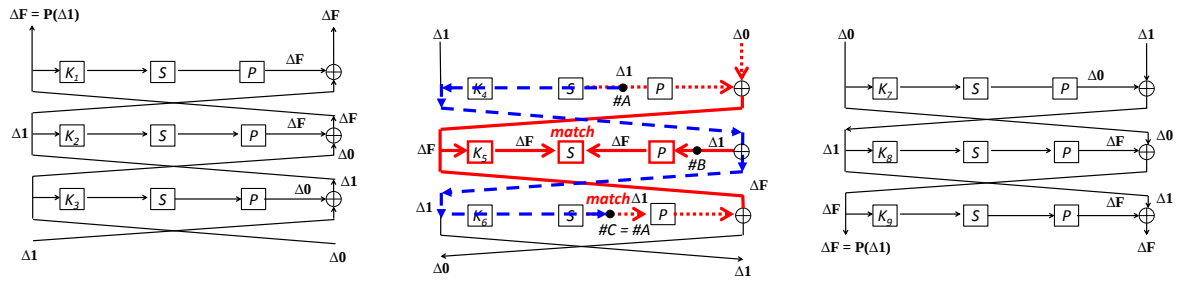
Knudsen and Rijmen proposed the known-key distinguishers on block ciphers [52]. They applied their attack on a reduced AES, as well as a 7-round Feistel cipher. Known-key distinguishers examine the randomness of a cipher when the key is known to the adversary. It is clear that if the adversary cannot find a distinguisher for a block cipher when the key is known, he definitely is not able to find any distinguisher when the key is secret. Studying known-key (or chosen-key) attacks is also important when the cipher is used as a building block of a hash function. In a hash function which is based on a block cipher, the key of the cipher is no longer private and is usually a random constant.

Later many researches have analysed block ciphers against known-key attacks. The known-key analysis of AES and Rijndael [26] with larger block size was studied in [80, 91]. The rebound attack was used to generate known-key distinguishers

on 8-round AES in [39]. In [84], the known-key and chosen-key attacks by the rebound technique were applied on the well-known block ciphers, Crypton [61], SAFER++ [68] and Square [28].

4.2.1 Known-key Attacks on Feistel-SP Ciphers

The rebound attack on generalised Feistel-SP ciphers is described in [93]. This attack is used to generate known-key distinguishers on Feistel-SP ciphers. Also if the cipher is used to build a hash function in the Matyas-Meyer-Oseas or Miyaguchi-Preneel modes, collision and half-collision attacks are possible by this technique. In [93] a rebound attack is introduced firstly on Feistel-SP ciphers with a 3-round inbound phase. Then it is extended to a 5-round inbound phase. The basic attack is shown in Figure 4.4, where the inbound phase consists of 3 rounds and the outbound phase includes 6 rounds (3 rounds in each forward and backward sides).



(a) Backward outbound **(b)** 3-round inbound **(c)** Forward outbound

Figure 4.4: Basic 9-round rebound attack on Feistel-SP networks

Henceforth, we denote a pair of differences for the left and the right hand halves as $(\Delta\mathbf{L}, \Delta\mathbf{R})$. The following notations are used to determine the truncated difference of a state or each half:

$\Delta\mathbf{0}$: A non-active state where all the differences are equal to zero.

$\Delta\mathbf{1}$: A state with only one byte with non-zero difference.

$\Delta\mathbf{F}$: A full-active state where all the bytes have non-zero differences.

The 3-round inbound phase is the first step of the basic 9-round rebound attack, which is shown in Figure 4.4b. The inbound phase starts with the truncated difference $(\Delta\mathbf{1}, \Delta\mathbf{0})$ and after 3 rounds ends with the truncated difference $(\Delta\mathbf{0}, \Delta\mathbf{1})$. To find the solutions of the inbound phase, the attacker first needs to prepare DDT of the S-boxes. Then for all 2^c possible differences in state $\#A$, the 4^{th} round permutation is applied and all the possible output differences are stored in table T. Now for every 2^c possible differences of state $\#B$, the 5^{th} round inverse permutation is applied and the corresponding full-active state is computed. By the help of the pre-computed DDT, a match is found between the difference coming from the state $\#B$ on one side of the S-boxes in round 5, and the table T entries on the other side. At this stage, the differences and their corresponding pairs of values are found for the bold red lines in Figure 4.4b. However, to make sure of having difference $\Delta\mathbf{0}$ on the sides of the inbound phase, the difference at position $\#C$ must be equal to the difference at $\#A$. Therefore, the attacker checks if for every difference at $\#A$, the possible pair of values results in the same difference at state $\#C$ through the broken blue lines in Figure 4.4b. After this second match, the attacher expects to find a solution for the inbound phase with $r.2^{2c}$ computations and $r.2^{2c}$ memory for table T. The outbound phase of the 9-round attack, propagates the solutions of the inbound phase 3 rounds forward and 3 rounds backward with probability 1. Although, at the first glance it seems the input and output truncated differences are fully active, which cover all the possible input/output differences. Closer look at the second round of both backward and forward outbound phases reveals that the right half input and left half output difference has only 2^c possible differences each. Because they both are the result of a permutation layer applied on a state with difference $\Delta\mathbf{1}$, so after the permutation there are only 2^c possible differences for each.

The 9-round rebound attack can be used to generate a known-key distinguisher when 2^{c+n} input differentials with the form of $(P(\Delta\mathbf{1}), \Delta\mathbf{F})$ generate 2^{c+n} output differentials with the form of $(P(\Delta\mathbf{1}), \Delta\mathbf{F})$. The rebound attack works better than the generic birthday attack if the complexity of the attack is less than the generic one, which is $r.2^{2c} < 2^{(n-c)/2}$. Therefore the following condition has to be met:

$$c < \frac{n - 2\log_2 r}{5}$$

The 3-round inbound phase is extended to 5 rounds in [93], depicted in Figure 4.5. Similar to the 3-round inbound phase, the attack starts with pre-computing DDTs for the S-boxes. Then for all the possible differences of state $\#A$, it finds differences at $\#B$ that they match at the S-box layer of round 2 (the bold lines in the figure). Difference of state $\#A'$ is equal to $\#A$, so exactly the same calculation is used to find a match over the S-box layer at round 4. Then the value of state $\#C$ is found from the values and differences found for $\#A$ and $\#A'$, indicated by the broken lines in the figure. Finally the consistency of differences at state $\#A$ and $\#A'$ is checked and the solutions of the inbound phase are found. Similar to the 3-round inbound phase, the complexity of the 5-round inbound phase is $r.2^{2c}$ time and $r.2^{2c}$ memory. In Section 4.3 we improve the 5-round inbound phase and find a solution for the inbound phase with 2^c time complexity (square root of the complexity from [93]).

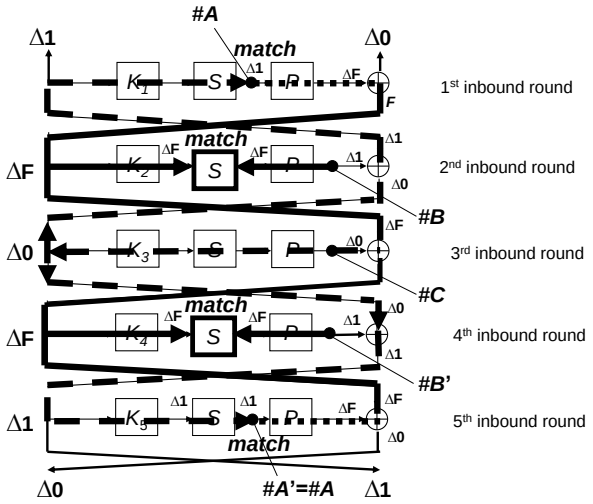


Figure 4.5: The 5-round inbound phase

4.3 Improved Rebound Attack on Feistel-SP Networks

In this section, we propose a new 5-round inbound phase where every solution is found with 2^c computations. The 5-round inbound phase is divided into four following sub-phases:

1. *First inbound phase*: A pair of values is obtained which follows the differential path of the 1^{st} and 2^{nd} rounds. Overall, 2^c solutions are generated.

2. *Second inbound phase*: Similar to the first inbound phase, a pair of values is computed for the 4^{th} and 5^{th} inbound rounds.

3. *Merging phase*: All possible solutions of the first and second inbound phases are combined in the 3^{rd} inbound round, in a way that they cancel each other differences at the 3^{rd} round.

4. *Validity check phase*: Regarding the active byte, the difference of the last round is checked.

The core of the improvement is the merging phase, in which we combine the results of the first and second inbound phases so that the n-bit match condition at the 3^{rd} round is always satisfied. More precisely, the merging phase first chooses a solution of the 1^{st} inbound phase from 2^c candidates. Then, it only chooses solutions of the second inbound phase which satisfy the n-bit condition (i.e. result in difference zero after the XOR addition). Finally, the validity check at the 5^{th} inbound round investigates the condition on the difference of the active byte. The success probability of the validity check is 2^{-c}. By iterating this phase for 2^c candidates of the first inbound phase, we will succeed with a negligible cost.

4.3.1 The Attack Procedure

For simplicity, we first assume that all S-boxes are identical and the parameters r and c satisfy $c \geq r + 1$. The S-box is also assumed to be designed to resist the differential cryptanalysis like the ones in AES or Camellia. The attack procedure is illustrated in Figure 4.6. The first inbound phase is denoted by the bold red

lines, and the second inbound phase by blue. Also, the merging and the validity check phases are depicted by the broken green and yellow lines, respectively. Instead of the permutation P in the 3rd round, we apply an inverse permutation P^{-1} to the input of the right hand half, and a permutation P to the output of the right hand half in round 3. It is clear this change does not affect the result of the Feistel network.

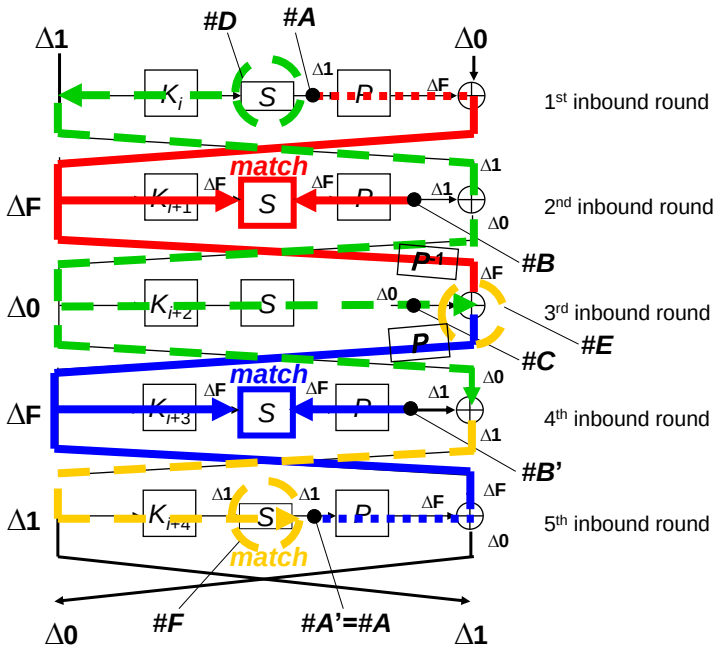


Figure 4.6: Improved 5-round inbound phase

The attack procedure is explained bellow (the pseudo-code is given in Algorithm 1):

First inbound phase (Steps 1 to 11 in Algorithm 1): Choose a difference at state $\#A$ (i.e. $\Delta\#A$), and compute $P(\Delta\#A)$, which is an input to the S-box layer in the 2nd inbound round. Then, choose all the possible differences at $\#B$ (i.e. $\Delta\#B$) such that the differential propagation through the active S-box in the first inbound round have solutions, it means $\exists x : S(x) \oplus S(x \oplus \Delta\#B) = \Delta\#A$. The number of such $\Delta\#B$s is approximately 2^{c-1}. For 2^{c-1} choices of $\Delta\#B$, compute $P^{-1}(\Delta\#B)$, which is an output of the

S-box layer in the 2^{nd} round. Check if all the S-boxes in the 2^{nd} round have solutions. If the check succeeds, for each of the possible solutions, compute the corresponding pairs of values at state $\#B$ (by applying P on the solutions of the 2^{nd} round S-boxes) and store them in table T_1. Moreover, compute the corresponding pairs of values at state $\#E$ at the right hand half input of the 3^{rd} round (by applying the 3^{rd} round sub-key addition and then P^{-1}), and store them in table T_1, as well. Table T_1 is expected to have 2^{c-1} entries.

Second inbound phase (Steps 12 to 13 in Algorithm 1): Set the same difference at state $\#A'$ as the difference at $\#A$ (i.e. $\Delta\#A' \leftarrow \Delta\#A$). Similar to the first inbound phase, compute 2^{c-1} solutions of the last two inbound rounds and store the resulting pairs of values at $\#E$ in table T_2.

Merging phase (Steps 14 to 18 in Algorithm 1): For 2^{c-1} solutions of the first two inbound rounds stored in T_1 and all solutions of the active S-box in the 1^{st} round at state $\#D$ (2 solutions on average), do as follows. Regarding only the active byte, compute the value up to the state $\#C$. If the computed value at state $\#E$ matches one of the entries in T_2, fix the solution for the last two rounds to this value. Go to the validity check with this value.

Validity check phase (Steps 19 to 21 in Algorithm 1): Regarding only the active byte, compute the value up to the output of the active S-box in the 5^{th} round ($\#F$). If the computed difference matches $\Delta\#A'$, the pair of values is a valid solution. Otherwise, go back to the merging phase.

4.3.2 Complexity Evaluation

Complexity of the first inbound phase is 2^c 1-round computations and 2^{c-1} states of memory (for table T_1). There are 2^{c-1} pairs of values for state $\#B$ in the second round, for every one of them 2 solutions are obtained for the active S-box in the first inbound round. Overall, the first inbound phase produces 2^c solutions with the cost of 2^c computations. The evaluation for the second inbound phase is exactly the same. The merging phase repeats for 2^c trials (from table T_1).

Algorithm 1 Improved 5-Round inbound phase

Require: DDTs for all the S-boxes

Ensure: A pair of values satisfying the truncated differential path of the inbound phase

1: Choose a difference at $\#A$ (i.e. $\Delta\#A$), and compute $P(\Delta\#A)$.
2: Choose all differences at $\#B$ (i.e. $\Delta\#B$s) such that $\exists x : S(x) \oplus S(x \oplus \Delta\#B) = \Delta\#A$. {The number of such $\Delta\#B$ is approximately 2^{c-1}}.
3: **for** 2^{c-1} choices of $\Delta\#B$ **do**
4: Compute $P^{-1}(\Delta\#B)$.
5: **if** all the S-boxes in the 2^{nd} round have solutions **then**
6: **for** each possible pair of solution X and X' (before the S-box) **do**
7: Store the corresponding pair of values at $\#B$, i.e. $< P(S(X)), P(S(X')) >$, in table $T_1[\#B]$
8: Store the corresponding pair of values at $\#E$, i.e. $< P^{-1}(X \oplus K_{i+1}), P^{-1}(X' \oplus K_{i+1}) >$, in table $T_1[\#E]$ {T_1 is expected to have 2^{c-1} entries}.
9: **end for**
10: **end if**
11: **end for**
12: Set $\Delta\#A' \leftarrow \Delta\#A$.
13: Compute 2^{c-1} solutions of the last two inbound rounds (similar to the first two inbound rounds) and store them in $T_2[\#B]$ and their corresponding pairs of values at state $\#E$ in table $T_2[\#E]$.
14: **for** each $< X, X' >$ in $T_1[\#E]$ **do**
15: **for** each solution of the active S-box at $\#D$ {2 solutions on average} **do**
16: Compute the pair of values $< Y, Y' >$ at state $\#C$, only for the active byte {with the help of the corresponding pair of values in $T_1[\#B]$}.
17: **if** $< X \oplus Y, X' \oplus Y' >$ exists in $T_2[\#E]$ **then**
18: Fix the solution for the last two rounds to this value
19: Compute the pair of values up to $\#F$, the output of the active S-box in the 5^{th} round {using the corresponding pair of values in $T_2[\#B]$}.
20: **if** the computed difference matches $\Delta\#A'$ **then**
21: **return** the pair of values as a solution of the inbound phase.
22: **end if**
23: **end if**
24: **end for**
25: **end for**

The match at state $\#E$ succeeds with probability 2^{-c}. However, considering 2^c solutions stored in T_2, we expect to find one match for each trial. Then, the results

are computed up to state $\#F$ for the validity check. The success probability of the validity check is 2^{-c}. Since the merging phase is iterated 2^c times, we expect to find one solution after the validity check. The merging phase and the validity check together require 2^c 5-round computations. Finally, one solution of the inbound phase is computed with 2^c 1-round $+ 2^c$ 1-round $+ 2^c$ 5-round $=$ 2^c 7-round computations. Note that the DDT is pre-computed and stored in memory before the attack. So, overall, if all the S-boxes are different the attacker needs almost $r.2^{2c}$ states of memory, otherwise 2^{2c} memory is enough.

At the beginning of Section 4.3.1 we assumed $c \geq r + 1$. However, the attack can also be applied to other cases with minor changes. Where $c = r$, in the first inbound phase (also the second inbound phase), there are only $2^{c-1} = 2^{r-1}$ pairs to examine. While the match for the 2^{nd} round S-box is possible with probability 2^{-r}. Therefore, there are not enough degrees of freedom to find a match for the S-box. However, the problem can be solved by running the first inbound phase for two different $\Delta\#A$. Hence, we can find a pair of values satisfying the first inbound phase differential with the complexity of 2^c computations (the same complexity as before). Also, in some of the other cases where $c < r$, we are still able to mount the attack. For example assume $r = 2c$. Here we consider every two S-boxes as a big S-box with the size of $2c$. Now the new parameters are $C = 2c$ and $R = r/2$. It means we activate two symbols rather than one everywhere we had difference $\Delta\mathbf{1}$ (e.g. state $\#A$). This gives us enough degrees of freedom to find a match of differences over the S-box layers in the first and second inbound phases. In this case, a pair of solutions are found for the inbound phase with $2^C = 2^{2c}$ time and memory complexity.

4.3.3 Distinguishing Attacks

The 5-round inbound phase can be used to find known-key distinguishers on Feistel-SP ciphers up to 11 rounds. The inbound phase includes the 5 middle rounds while the outbound phase consists of 3 backward and forward rounds similar to Figures 4.4a and 4.4c, respectively. The 11-round truncated differential

path is as follows:

$$\textit{backward outbound:}(P(\Delta\mathbf{1}), \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{1}, P(\Delta\mathbf{1})) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{1}) \rightarrow (\Delta\mathbf{1}, \Delta\mathbf{0})$$

$$\textit{5-round inbound:}(\Delta\mathbf{1}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{F}, \Delta\mathbf{1}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{F})$$

$$\rightarrow (\Delta\mathbf{F}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{1}, \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{1})$$

$$\textit{forward outbound:}(\Delta\mathbf{0}, \Delta\mathbf{1}) \rightarrow (\Delta\mathbf{1}, \Delta\mathbf{0}) \rightarrow (P(\Delta\mathbf{1}), \Delta\mathbf{1}) \rightarrow (P(\Delta\mathbf{1}), \Delta\mathbf{F})$$

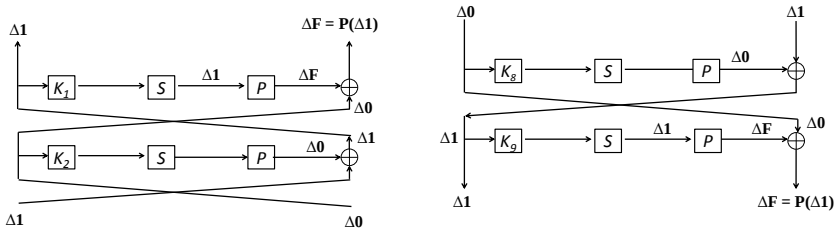
The solution of the inbound phase follows the outbound truncated differential with probability 1. It allows us to find a pair of plaintexts whose difference is $(P(\Delta\mathbf{1}), \Delta\mathbf{F})$ and a pair of ciphertexts with the difference $(P(\Delta\mathbf{1}), \Delta\mathbf{F})$. The complexity of the attack depends only on the complexity of the inbound phase, which is 2^c time and 2^{2c} memory. For example, assume a 128-bit Feistel-SP cipher with the following parameters: $N = 128$, $n = 64$, $c = 8$ and $r = 8$, where the S-boxes are identical. Our known-key distinguisher on 11 rounds is successful with 2^8 time and 2^{16} memory.

4.3.4 Attacks on the Hashing Modes

Recall from Chapter 2 (Section 2.2.3) block ciphers can be used to build hash functions. In this section, we examine security of the hash functions which are built from Feistel-SP ciphers according to the Matyas-Meyer-Oseas or Miyaguchi-Preneel schemes. Given a block cipher E_K with a key K, the compression function in the Matyas-Meyer-Oseas scheme computes h_i by $h_i = E_{h_{i-1}}(M_i) \oplus M_i$ for the message M_i and the previous chaining value h_{i-1}. While the Miyaguchi-Preneel scheme computes h_i by $h_i = E_{h_{i-1}}(M_i) \oplus M_i \oplus h_{i-1}$. When we apply the rebound attack on the compression function, the chaining value h_{i-1} is assumed a fixed constant, which is the key for the block cipher. Therefore, the differential attack on both the Matyas-Meyer-Oseas and Miyaguchi-Preneel schemes acts similarly, because there is no difference in the key (i.e. h_{i-1}).

9-round Collision Attack

To apply the collision attacks, the attacker has to repeat the inbound phase to generate more than one starting point for the outbound phase. Then he finds a collision by cancelling the plaintext difference with the ciphertext difference. We use the 5-round inbound phase explained in Section 4.3.1 to mount a collision attack on 9 rounds of a Feistel-SP based compression function. The outbound phase consists of 2 forward and 2 backward rounds (see Figure 4.7).



(a) Backward outbound phase **(b)** Forward outbound phase

Figure 4.7: The outbound phase of the 9-round collision attack

The truncated differential path for the whole 9 rounds is as follows:

$$\textit{2-round backward outbound:}(\Delta\mathbf{1}, P(\Delta\mathbf{1})) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{1}) \rightarrow (\Delta\mathbf{1}, \Delta\mathbf{0})$$

$$\textit{5-round inbound:}(\Delta\mathbf{1}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{F}, \Delta\mathbf{1}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{F})$$

$$\rightarrow (\Delta\mathbf{F}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{1}, \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{1})$$

$$\textit{2-round forward outbound:}(\Delta\mathbf{0}, \Delta\mathbf{1}) \rightarrow (\Delta\mathbf{1}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{1}, P(\Delta\mathbf{1}))$$

The input truncated difference is $(\Delta\mathbf{1}, P(\Delta\mathbf{1}))$ and so the output difference. After the feed-forward operation on the cipher (to generate the compression function) the input and the output differences are XORed. Therefore, if we find the same differential for the input and the output, they cancel each other and the compression function output difference will be zero. Hence, the pair of values conforming to the input differential makes a collision for the compression function. The left-hand half differentials (i.e. $\Delta\mathbf{1}$) cancel each other with probability

2^{-c}. The right-hand half differences are fully-active differences, which both are the result of transferring a state with the difference $\Delta\mathbf{1}$ through the linear transformation P (i.e. $P(\Delta\mathbf{1})$). So for cancelling these differences it is enough to find a match for the states with the difference $\Delta\mathbf{1}$ before the P transformation. The match is found with probability 2^{-c}. Thus, the probability of finding the same differences for the input and the output is 2^{-2c}. To find at least one collision the attacker has to repeat the inbound phase 2^{2c} times, whereas the inbound phase complexity is 2^c computations (and 2^{2c} memory). Overall, the collision attack on the 9-round compression function is successful by 2^{3c} computations and 2^{2c} memory.

11-round Half-Collision Attack

The 11-round distinguisher, described in Section 4.3.3, can be easily used to generate a half-collision attack on the hashing mode. The input and output differences are of the form of $(P(\Delta\mathbf{1}), \Delta\mathbf{F})$. The left-hand half difference is full-active and finding a match for this half has a very low probability. However, the right-hand half difference is the result of the linear transformation P on a state with difference $\Delta\mathbf{1}$, so a match is found for this half with probability 2^{-c}. Hence, using the 5-round inbound phase starting points, a collision for the left-hand half of the compression function is found with the complexity of 2^{2c} computations and 2^{2c} memory. Note that the generic attack complexity for finding half-collision is $2^{N/4}$ computations in an N-bit ideal compression function. For example, for a compression function built from a Feistel-SP cipher with parameters, $N = 128$, $n = 64$, $c = 8$ and $r = 8$, the generic complexity of finding a half-collision is 2^{32}. While, our attack complexity is 2^{16} time and 2^{16} memory.

4-Sum Distinguishers

A k-sum is a type of zero-sum distinguisher, where the XOR addition of the k messages' outputs is zero. A special form of k-sum is the 4-sum distinguisher. A 4-sum distinguisher on a compression function F, finds four messages x_1, x_2, x_3 and x_4 such that $F(x_1) \oplus F(x_2) \oplus F(x_3) \oplus F(x_4) = 0$. According to the theory, 4-sum messages can be found by $2^{N/4}$ computations, for an N-bit algorithm.

However, the birthday attack is the best known practical generic attack so far, which finds 4-sums after $2^{N/3}$ computations [105]. The 4-sum distinguishers can be used to form boomerang attacks on hash functions [48, 55]. Moreover, they are used in different applications such as the attack on a random oracle instantiation in [59], and the attack on a signature scheme in [105].

We explain here that the 11-round rebound attack of Section 4.3.3 can be used to form a 4-sum distinguisher on the full state of the compression function. In the 11-round rebound attack, each solution of the inbound phase produces a pair of inputs (and their corresponding outputs) with the truncated difference of $(P(\Delta\mathbf{1}), \Delta\mathbf{F})$. Let the pair of values $< x_1, x_2 >$ be a solution of the inbound phase, and $< y_1, y_2 >$ be the corresponding output pair after the feed-forward. Then, $y_1 \oplus y_2 = (P(\Delta\mathbf{1}), \Delta\mathbf{F})$ (i.e. $((P(\Delta\mathbf{1}), \Delta\mathbf{F}) \oplus (P(\Delta\mathbf{1}), \Delta\mathbf{F}) = (P(\Delta\mathbf{1}), \Delta\mathbf{F}))$). After running the inbound phase one more time, another pair of values $< x'_1, x'_2 >$ is found, where its corresponding output pair is $< y'_1, y'_2 >$. The 4-sum of these input values is the second-order difference $(y_1 \oplus y_2) \oplus (y'_1 \oplus y'_2)$, and it becomes zero if $y_1 \oplus y_2 = y'_1 \oplus y'_2$. Since $y_1 \oplus y_2$ takes 2^{c+n} possibilities, using the birthday paradox, the 4-sum distinguisher can be generated with $2^{(c+n)/2}$ solutions of the inbound phase.

4.4 Application to Camellia-128

In this section, we use the proposed method on generic Feistel-SP ciphers to attack Camellia [3]. Camellia is not a plain Feistel-SP cipher; because the P operation is not MDS, and there are FL and whitening layers in the design. So, the attack needs several modifications to succeed. We evaluate Camellia which includes FL and whitening layers, with the key size of 128 bits.

Most of the analysis on Camellia do not consider FL-layer and the whitening layer in their evaluations. For Example, the impossible differential attack is used in [67] to attack 12 rounds of Camellia-128. From those who consider the FL-layer, the impossible differential attacks are applied on 10 rounds of Camellia-128 in [60, 64, 65]. An impossible differential attack on Camellia-128 reduced to 11 rounds is presented in [5]. Also, the meet-in-the middle attack is used to attack 10 rounds of Camellia-128 in [66]. Table 4.1 shows the complexity of the attacks on

Camellia-128 including our results. Note that we mostly attack Camellia on the hashing mode. While, the other key-recovery attacks do not indicate a collision attack on the hashing mode faster than the birthday attack.

Table 4.1: Complexity of the attacks on Camellia-128 and its hashing modes

Target	Type of Attack	rounds	Time	Reference
Block cipher	Impossible differential	10	$2^{123.5}$	[60]
	Impossible differential	10	2^{118}	[65]
	Impossible differential	10	2^{120}	[64]
	Impossible differential	11	$2^{123.8}$	[5]
	Meet-in-the-middle	11	$2^{121.5}$	[66]
	Chosen-key distinguisher	11	2^{16}	Section 4.4.2
Hash function	4-sum	7	2^{32}	Section 4.4.3
Compression function	4-sum	9	2^{40}	Section 4.4.3
	Collision	9	2^{48}	Section 4.4.3
	Half-collision	9	2^{16}	Section 4.4.3

4.4.1 Camellia Description

Camellia [3] is a Japanese block cipher designed by NTT and Mitsubishi Electric Corporation, and was specified as an international standard by different organisations such as, ISO/IEC [46], NESSIE [83], and CRYPTREC [25]. Camellia supports three key sizes; 128, 192, and 256 bits, and the block size is always 128 bits. Here we only explain the version with 128-bit key, referred to as Camellia-128.

The 128-bit master key K is used to generate eighteen 64-bit round sub-keys $k_1, \ldots, k_{18}$, four 64-bit whitening sub-keys $kw_1, \ldots, kw_4$, and four 64-bit sub-keys $kl_1, \ldots, kl_4$ for the FL layer. The 128-bit input plaintext M splits into two halves for the Feistel structure. L_r and R_r $(0 \leq r \leq 18)$ are the 64-bit left and right halves of the state in round r. After the input whitening operation the input state (L_0, R_0) is computed as, $L_0 \| R_0 \leftarrow M \oplus (kw_1 \| kw_2)$. Also, there is an output whitening layer which applies the XOR addition with the sub-keys $kw_3 \| kw_4$ to the output state. The round function F is iterated for 18 rounds,

while FL and FL^{-1} are applied to the state every 6 rounds. The block cipher operation is formalized as follows:

$L_0 \| R_0 \leftarrow M \oplus (kw_1 \| kw_2)$
for $i = 1, 2, ..., 18$ **do**
  $L_r = R_{r-1} \oplus F(L_{r-1}, k_r)$
  $R_r = L_{r-1}$
  if $r = 6$ **or** $r = 12$ **then**
    $L_r = FL(L_r, kl_{r/3-1})$
    $R_r = FL^{-1}(R_r, kl_{r/3})$
  end if
end for
$L_{18} \| R_{18} \leftarrow (L_{18} \| R_{18}) \oplus (kw_3 \| kw_4)$

Key Schedule

Here we give a brief description of the key schedule where the key size is 128 bits (the details are explained in Appendix A). The key schedule takes 128-bit master key K as the input and generates 128-bit register K_A. To compute K_A, K is encrypted through 2 rounds of Camellia (the sub-keys are pre-specified constants), then the output value is XORed with K and again encrypted 2 rounds. The 64-bit required sub-keys are generated from K and K_A. It is worth to mention that for the known-key attacks, sub-keys are given as constants, thus the key schedule process does not change the results. We explain later for chosen-key attacks, the sub-keys are chosen in a way that they conform to the key-schedule.

Round Function

The round function F uses the SPN structure and consists of a sub-key addition followed by a S-box layer and a permutation layer (as shown in Figure 4.8). 4 different S-boxes are used in the S-box layer which are designed to be resistant against differential and linear cryptanalysis. The DDTs for these S-boxes have the same property as the one for AES. The permutation layer applies a linear function P on the outputs of the S-boxes. Let $z_1 \| z_2 \| \cdots \| z_8$ be the 64-bit output

of the S-box layer. The output of the P function, $z'_1\|z'_2\|\cdots\|z'_8$, is computed as follows:

$$z'_1 = z_1 \oplus z_3 \oplus z_4 \oplus z_6 \oplus z_7 \oplus z_8, \qquad z'_2 = z_1 \oplus z_2 \oplus z_4 \oplus z_5 \oplus z_7 \oplus z_8,$$
$$z'_3 = z_1 \oplus z_2 \oplus z_3 \oplus z_5 \oplus z_6 \oplus z_8, \qquad z'_4 = z_2 \oplus z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7,$$
$$z'_5 = z_1 \oplus z_2 \oplus z_6 \oplus z_7 \oplus z_8, \qquad z'_6 = z_2 \oplus z_3 \oplus z_5 \oplus z_7 \oplus z_8,$$
$$z'_7 = z_3 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_8, \qquad z'_8 = z_1 \oplus z_4 \oplus z_5 \oplus z_6 \oplus z_7.$$

Note the branch number of P is 5, so it is not a MDS transformation. Although it does not satisfy the assumption that we made for Feistel-SP ciphers in Section 4.1.2, we show later how with some changes we can apply the attack on Camellia.

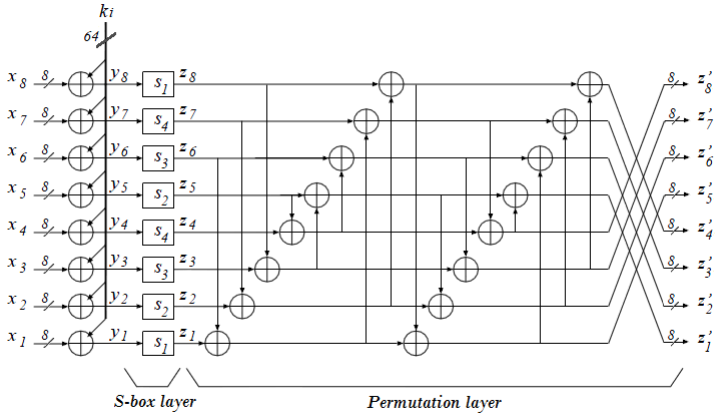


Figure 4.8: Camellia round function F [3]

FL and FL^{-1} Functions

The FL function takes a 64-bit value $X_L\|X_R$ and a 64-bit sub-key $kl_L\|kl_R$ as the input and produces a 64-bit output value $Y_L\|Y_R$ by the following computations:

$$Y_R = \big((X_L \cap kl_L) \lll 1\big) \oplus X_R, \qquad Y_L = (Y_R \cup kl_R) \oplus X_L.$$

where $\cap$, $\cup$, and $\lll 1$ are the logical AND, OR, and a left rotation by 1 bit, respectively.

The FL^{-1} function is designed such that $FL^{-1}(FL(x,k),k) = x$. So if the 64-bit value $Y_L \| Y_R$ is the input to FL^{-1}, the output $X_L \| X_R$ is generated as follows (with the same sub-key):

$$X_L = (Y_R \cup kl_R) \oplus Y_L, \qquad X_R = \big((X_L \cap kl_L) \lll 1\big) \oplus Y_R.$$

4.4.2 The Rebound Attack on Camellia-128

In this section, the known-key and chosen-key attacks on Camellia-128 block cipher are described. The chosen-key distinguisher was firstly proposed by Biryukov and Nikolić [15], and later discussed in many other works [14, 31, 84]. These distinguishers are able to choose the key value. The attacker may choose sub-keys separately, however the chosen sub-keys must follow the key schedule of the cipher. An important application of chosen-key attacks is for the compression functions designed based on block ciphers. Attacking a compression function, the adversary can choose the initial value of his own choice, which corresponds to the key in the block cipher.

First we explain how we modify the proposed technique to attack Camellia, regarding the small branch number of P transformation and the existence of FL and FL^{-1} functions.

Dealing with the Small Branch Number of P

The P function in Camellia maps an 8-byte value to another 8-byte value linearly. To apply the rebound attack we have assumed that the Feistel-SP cipher uses a MDS permutation. However the permutation in Camellia is not MDS. The problem occurs when we want to find a match between two full-active states over a S-box. More precisely, if we start with differences of form $\Delta\mathbf{1}$ at positions $\#A$ and $\#B$ (in Figure 4.6), and propagate them through P and P^{-1} functions, we might not get similar active positions before and after the S-box. To overcome this problem, the position of the active bytes has to be chosen carefully. The

following conditions on active byte positions must be met to keep the consistency with the other part of the inbound phase differential path:

1. Active byte positions for $\#A$ and $\#B$ must be identical. So, the match of differences can be found in Step 2 of Algorithm 1.

2. The active byte positions of $P(\Delta\#A)$ and $P^{-1}(\Delta\#B)$ must be identical. Thus, the match of differences can be found in Step 5 of Algorithm 1. Note that it is not necessary to activate all the bytes in the states.

No single active byte position, satisfy the second condition. We solve the problem by activating two bytes at states $\#A$ and $\#B$. There are $\binom{8}{2} = 28$ possibilities for the position of two active bytes. Table 4.2 determines which two positions meet the above conditions. There are eight possible pair of positions for the active bytes. We choose positions 5th and 7th and activate these bytes instead of just one at states $\#A$ and $\#B$. Hereafter, we use the notation $\Delta\mathbf{10100000}$ to represent that only 5th and 7th bytes have non-zero differences.

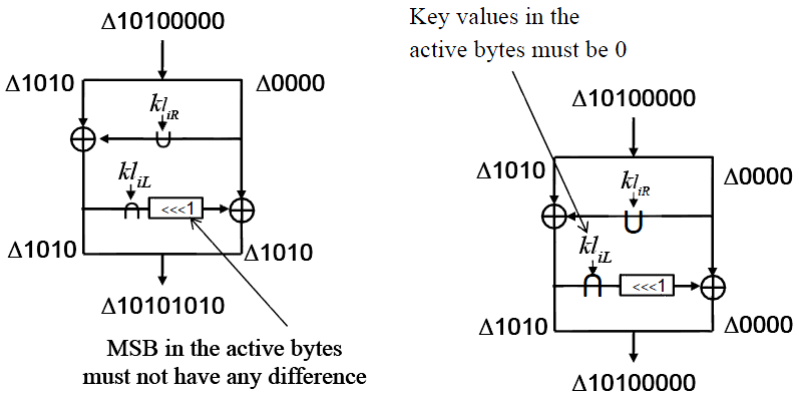
Table 4.2: How two different active byte positions satisfy the conditions

	0th	1st	2nd	3rd	4th	5th	6th	7th
0th		×	√	×	×	×	×	×
1st			×	√	×	×	×	×
2nd				×	×	×	×	×
3rd					×	×	×	×
4th						√	√	√
5th							√	√
6th								√

Dealing with FL and FL^{-1} Functions

In a differential path FL and FL^{-1} are applied every 6 rounds to the state and propagate the difference of each half state. Therefore, they cannot be inside the inbound phase. We choose the starting round of our attacks in a way that these functions are placed immediately after the inbound phase. The output difference of both 3-round and 5-round inbound phase is of the form of $(\Delta\mathbf{0}, \Delta\mathbf{10100000})$. Where the left-hand difference $\Delta\mathbf{0}$ goes to the FL function and the right-hand

different $\Delta\mathbf{10100000}$ goes to FL^{-1}. It is obvious that $\Delta\mathbf{0}$ results in $\Delta\mathbf{0}$ through FL function. Hence, we need to examine the right-hand differential propagation through FL^{-1}.



(a) Known-key scheme **(b)** Chosen-key scheme

Figure 4.9: Truncated differential propagation through FL^{-1}

Both known-key and chosen-key distinguishers aim to keep the number of active bytes low. Figure 4.9 shows the truncated differential propagation through FL^{-1}. In the known-key setting, in order to prevent the difference from propagating through 1-bit left rotation (i.e. $\lll 1$ operation), we should avoid activating the most significant bit (MSB) in each active byte. This reduces the degrees of freedom at the time of selecting differences (we have enough freedom to proceed in our attacks, though). Also instead we may apply the attack on a weak set of the keys while not avoiding the difference at the MSB of the active bytes. If in the key kl_{iL} the MSB of each byte corresponding to the active byte is 0, the logical AND between the key and the state results in difference 0. So the 1-bit left rotation does not propagate the difference to another byte. Such weak keys exist with probability 2^{-2}, and the number of weak keys is $2^{128-2} = 2^{126}$. This idea helps us in the chosen key scheme, where the distinguisher chooses the key value. Choosing one sub-key value is trivially done with complexity 1 for any key value. This is because kl_i is a part of K or K_A. If it is a part of K, the distinguisher directly chooses the value. If it is a part of K_A, the distinguisher firstly chooses K_A and then inverts it to K through the key schedule.

The 11-round Distinguisher

The 5-round inbound phase (explained in Section 4.3.1) is used to generate a chosen-key distinguisher on Camellia-128 reduced to 11 rounds. As mentioned above, we need to activate 2 S-boxes to manage the permutation layer, which is similar to having a big S-box of size $2c = 16$ bits. Hence, one solution of the inbound phase is obtained by 2^{16} computations and 2^{18} (i.e. 4×2^{16}) memory. The 5-round inbound phase can be followed by 3-round backward and 3-round forward outbound phases with probability one. The 11-round truncated differential path is,

3-round backward outbound:

$$(P(\Delta\mathbf{2}), \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{2}, P(\Delta\mathbf{2})) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{2}) \xrightarrow{FL-layer} (\Delta\mathbf{0}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{0})$$

5-round inbound:

$$(\Delta\mathbf{2}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{F}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{F}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{2})$$

3-round forward outbound:

$$(\Delta\mathbf{0}, \Delta\mathbf{2}) \xrightarrow{FL-layer} (\Delta\mathbf{0}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{0}) \rightarrow (P(\Delta\mathbf{2}), \Delta\mathbf{2}) \rightarrow (P(\Delta\mathbf{2}), \Delta\mathbf{F})$$

For simplicity we use $\Delta\mathbf{2}$ to show the 2-byte active differences at positions 5 and 7. In addition to the immediate FL-layer after the inbound phase, the distinguisher should control another FL-layer in the backward outbound phase. The FL-layer is placed after one backward outbound round. The form of the difference going to the inverse of this layer is $(\Delta\mathbf{0}, \Delta\mathbf{10100000})$. Therefore, we need to analyze the differential propagation of $(FL^{-1})^{-1}(\Delta\mathbf{10100000})$, which is equivalent to $FL(\Delta\mathbf{10100000})$. As Figure 4.10 shows, the analysis is similar to the above where the difference goes to FL^{-1}.

Therefore, if the distinguisher chooses the values of 2 active bytes of the sub-key equal to zero, the form of the output difference does not change from $\Delta\mathbf{10100000}$. For this reason, the distinguisher needs to control two bytes of kl_i, which is a part of either K or K_A. To make the both FL-layers work as expected, the distinguisher needs to fix two bytes of kl_1 and kl_4 to zero. From the key

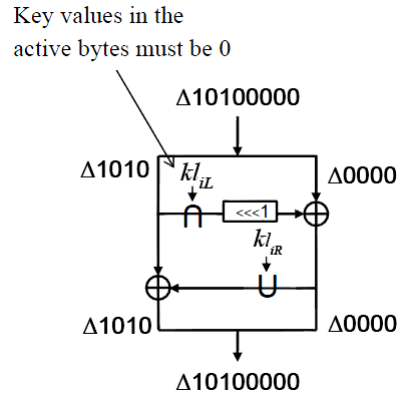


Figure 4.10: Chosen-key truncated differential propagation through FL

schedule (see Appendix A) we know that kl_1 is the left 64 bits of $(K_A \lll 30)$, and kl_4 is the right 64 bits of $(K \lll 77)$. Therefore, it is necessary to control two bytes of K_A and two bytes of the master key K. So, we fix the determined 2 bytes of K to zero and set the others randomely; then check if the expected two bytes in K_A are also valued by zero. The success probability of this event is 2^{-16}. We implemented the chosen-key search procedure, and confirmed that such keys are generated with approximately 2^{16} key schedule computations. The complexity of the 11-round chosen-key distinguisher on Camellia-128 depends on the complexity of the inbound phase. Therefore, it is successful with 2^{16} computations and requires 2^{18} memory.

4.4.3 Attacks on the Camellia-128 Hashing Modes

The rebound attacks on the hashing mode rely on 3-round and 5-round inbound phases. The 3-round inbound phase for Camellia is based on the one presented in [93]. However, we need to activate 2 S-boxes for Camellia (instead of 1), also we apply some optimizations to make the attack complexity 2^c, while the one proposed in [93] needs 2^{2c} computations.

Optimized 3-Round Inbound Phase

Figure 4.11 depicts the 3-round inbound phase for Camellia. Since the permutation of the Camellia round function is not MDS, we need to activate two S-boxes

at state $\#A$ and $\#B$ and then match over the second round S-box.

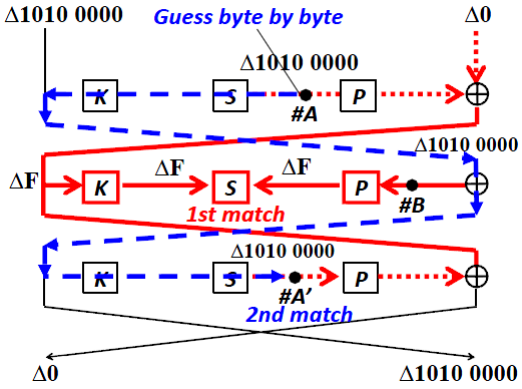


Figure 4.11: 3-round inbound phase for Camellia

To find the solutions of the 3-round inbound phase, first the attacker chooses the differences of state $\#A$ and $\#B$ in a way that they match over the S-box layer of the 2^{nd} round. After finding a match, the corresponding pair of values is found for the 2^{nd} round (displayed by the bold red line in the figure). So, the deferences are known at states $\#A$ and $\#A'$, but the values are not clear. Then, the attacker searches for the value of each active byte at state $\#A$ that satisfies the difference of the same byte position at state $\#A'$ (the broken blue line in the figure). The attacker finds the values of each byte one at a time. Algorithm 2 describes the attack procedure. Steps 1 to 7 finds the differences for states $\#A$, $\#A'$ and $\#B$. Then finds the values for the active bytes just for the 2^{nd} round, which is shown by the bold red line in the figure. Complexity of these steps are 2^8 computations all together. Steps 8 to 14 compute the values for byte 5 at states $\#A$ and $\#A'$, which requires 2^7 1-byte computations. The same method is used from Step 15 to 20 to find the pair of values at byte 7, with 2^7 1-byte computations. Finally a pair of values is found for the 3-round inbound phase with 2^8 computations (we need about 2^{18} state of memory to keep DDTs of all the S-boxes). Note that once a solution is found, at Step 21, we can generate up to 2^{48} pairs of values with no extra effort and just by choosing different values for the six non-active bytes.

Algorithm 2 Optimized 3-Round inbound phase for Camellia-128

Require: DDTs for all the S-boxes
Ensure: A pair of values satisfying the truncated differential path of the inbound phase

1: **for** 2^8 differences at $\#A$ (i.e. $\Delta\#A$){If it is a known-key attack, activating the MSB must be avoided} **do**
2: Compute $P(\Delta\#A)$
3: Choose a difference at $\#B$ (i.e. $\Delta\#B$s) and compute $P^{-1}(\Delta\#B)$ {If it is a known-key attack, activating the MSB must be avoided}
4: **if** all the S-boxes in the 2^{nd} round have solutions **then**
5: compute the corresponding pair of values for the whole state before and after the S-box at the 2^{nd} round {i.e. the red line in Figure 4.11}
6: **end if**
7: **end for**
8: Set $\Delta\#A' \leftarrow \Delta\#A$.
9: **for** All 2^7 possible pair of values for the 5^{th} byte at $\#A$ **do**
10: Compute the pair of values only for the 5^{th} byte up to state $\#A'$.
11: **if** the difference of the pair of values at byte 5 is equal to $\Delta\#A'$ **then**
12: Fix the pair of values for the 5^{th} byte in the whole 3 rounds
13: **end if**
14: **end for**
{do the same for byte 7}
15: **for** All 2^7 possible pair of values for the 7^{th} byte at $\#A$ **do**
16: Compute the pair of values only for the 7^{th} byte up to state $\#A'$.
17: **if** the difference of the pair of values at byte 7 is equal to $\Delta\#A'$ **then**
18: Fix the pair of values for the 7^{th} byte in the whole 3 rounds
19: **end if**
20: **end for**
21: Choose random values for 6 non-active bytes at $\#A$ and $\#A'$ {These values can take any random values. So they are used to increase the degrees of freedom by producing more solutions for the inbound phase, if required}
22: **return** the pair of values as a solution for the inbound phase

4-Sum Distinguishers

The known-key distinguisher is used to find a 4-sum distinguisher on the hash function built based on the Camellia-128 block cipher. The hash function is reduced to 7 rounds including the FL-layer. We use the 3-round inbound phase as the core of the 7-round trail, while the FL-layer is placed exactly after the

inbound phase. Two backward and two forward rounds propagate the inbound phase differential as,

2-round backward outbound:$(\Delta\mathbf{2}, P(\Delta\mathbf{2})) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{0})$

3-round inbound:$(\Delta\mathbf{2}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{F}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{2})$

2-round forward outbound:$(\Delta\mathbf{0}, \Delta\mathbf{2}) \xrightarrow{FL-layer} (\Delta\mathbf{0}, \Delta\mathbf{4}) \rightarrow (\Delta\mathbf{4}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{4}, P(\Delta\mathbf{4}))$

As shown in Figure 4.9a FL^{-1} propagates the input difference $\Delta\mathbf{10100000}$ to $\Delta\mathbf{10101010}$, for simplicity we use the notation $\Delta\mathbf{4}$ to show this difference with 4 active bytes. In the hashing mode, the output is the result of the feed-forward, which its difference is $\Big(\Delta\mathbf{2} \oplus \Delta\mathbf{4}, P(\Delta\mathbf{2}) \oplus P(\Delta\mathbf{4})\Big)$. This difference is in the space of $(\Delta\mathbf{4}, P(\Delta\mathbf{4}))$. We explained in Section 4.3.4 that if the output differences of two pair of massages collide, they form a 4-sum. Hence, we should find a collision between 8 bytes (i.e. 4 bytes to match $\Delta\mathbf{4}$, and 4 bytes for $P(\Delta\mathbf{4})$). So, the 4-sum distinguisher should generate at least $2^{(4+4)\times 8/2} = 2^{32}$ pairs which are following the differential path to find 4 messages that sum up to zero. Note that as mentioned in Section 4.3.4, the best known attack is the birthday paradox which finds 4-sum with $2^{128/3} \backsimeq 2^{42.6}$ computations.

The chosen-key distinguisher is also used to form a 4-sum distinguisher on 9 rounds of the compression function. The 3-round inbound phase is followed 3 rounds forward and backward while the FL-layer is applied on the output of the inbound phase. The 9-round differential trail is as follows:

3-round backward outbound:$(P(\Delta\mathbf{2}), \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{2}, P(\Delta\mathbf{2})) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{0})$

3-round inbound:$(\Delta\mathbf{2}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{F}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{2})$

3-round forward outbound:$(\Delta\mathbf{0}, \Delta\mathbf{2}) \xrightarrow{FL-layer} (\Delta\mathbf{0}, \Delta\mathbf{2})$

$\rightarrow (\Delta\mathbf{2}, \Delta\mathbf{0}) \rightarrow (P(\Delta\mathbf{2}), \Delta\mathbf{2}) \rightarrow (P(\Delta\mathbf{2}), \Delta\mathbf{F})$

The output difference is of the form of $(P(\Delta\mathbf{2}), \Delta\mathbf{F})$. So, the 4-sum distinguisher needs to generate $2^{(2+8)\times 8/2} = 2^{40}$ pairs of values to succeed. This complexity is still less than the best known birthday attack.

Collision Attacks

The 9-round chosen-key differential path that we use above to generate the 4-sum distinguisher, can be the path for a half-collision attack on the compression function. We just find collisions for the left-hand half of the state. The output difference of the left-hand half is $\Delta\mathbf{2}$ and we want to cancel it by the feed-forward which has the same two active bytes. The probability of having the same difference in the output and the feed-forward is 2^{-16}. Therefore, by generating 2^{16} solutions for the inbound phase we are able to find a collision. Note that the inbound phase needs 2^{18} states of memory, as well.

We can find a collision attack on the whole state using the 5-round inbound phase (i.e. used for the 11-round chosen-key distinguisher in Section 4.4.2). We follow the inbound phase 2 rounds backward and then 2 rounds forward as bellow,

2-round backward outbound:

$$(\Delta\mathbf{2}, P(\Delta\mathbf{2})) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{2}) \xrightarrow{FL-layer} (\Delta\mathbf{0}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{0})$$

5-round inbound:

$$(\Delta\mathbf{2}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{F}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{F}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{F}) \rightarrow (\Delta\mathbf{0}, \Delta\mathbf{2})$$

2-round forward outbound:

$$(\Delta\mathbf{0}, \Delta\mathbf{2}) \xrightarrow{FL-layer} (\Delta\mathbf{0}, \Delta\mathbf{2}) \rightarrow (\Delta\mathbf{2}, \Delta\mathbf{0}) \rightarrow (\Delta\mathbf{2}, P(\Delta\mathbf{2}))$$

So, if the feed-forward cancels the output difference of the whole state, with the difference $(\Delta\mathbf{2}, P(\Delta\mathbf{2}))$, we find a collision for 9 rounds of the compression function. These differences cancel each other with the probability of $2^{-(2+2)\times 8} = 2^{-32}$. Thus we need to generate 2^{32} solutions for the inbound phase to get a collision. However, each solution of the inbound phase is generated after 2^{16} computations and needs 2^{18} states of memory. Therefore, overall a collision is found after 2^{48} computations and 2^{18} memory.

4.4.4 Experiments and Generated Data

To verify the attacks, we implemented the chosen-key 5-round inbound phase. First of all, a valid key is chosen in a way that the FL-layers before and after the inbound phase do not propagate the differences to more bytes. By the experiments, we find a key with this property as expected after 2^{16} computations on average. Then, the chosen keys are used to find solutions of the new 5-round inbound phase. We implemented the 5-round inbound phase as explained in Algorithm 1. We find most of the solutions in less than 2^8 computations in our experiments. This interesting observation indicates that the real complexity of the attack is smaller than the theoretical estimate (i.e. 2^{16} computations). Table 4.3 shows some sample solutions that we have found by the experiments.

Table 4.3: Sample solutions of the 5-round inbound phase for Camellia-128 (in hexadecimal format)

	Left value L	Left difference ΔL	Right value R	Right difference ΔR
Input	BD403C2B535BBA20	3800100000000000	EC24733F6E3CF494	0000000000000000
Output	3B4A548FB81FEB7E	0000000000000000	8F2E46DA8E05B6AA	2D001C0000000000
Chosen key: 0A4294A1CC26E3FE50EB806F005C818				
Input	2F76D7BDB6D247F3	C100690000000000	1687F396523AF59D	0000000000000000
Output	83C827D878A9A6B8	0000000000000000	986EFFF44190697D	B2008F0000000000
Chosen key: 95E3387988B2DD056E556A03E8065052				
Input	F73E7FD82B7F8963	4500860000000000	2A53B729FFADF353	0000000000000000
Output	9BBC3A1527ABFD70	0000000000000000	735C0A12250854DA	9800F10000000000
Chosen key: F90C1C05F009F136FC84540210006077				
Input	97BB49870767052B	F7004A0000000000	B74F131532BBDA46	0000000000000000
Output	7602FC9D42E5E6D3	0000000000000000	6C98EED9141FB49E	8500190000000000
Chosen key: 791C72399212333DDCF4A6041000682F				
Input	EBADF14A23994789	B5002B0000000000	6BD5743F3BE0C424	0000000000000000
Output	A1B653F0331053E	0000000000000000	D39426289D088187	9000020000000000
Chosen key: F2D4C755AB60712256070301C803002C				

4.5 Conclusion

In this chapter, we have first revisited the known-key attacks on generic Feistel-SP ciphers using the rebound attack. Our main contribution is a new 5-round inbound phase, which improves the efficiency of the attack. So, it can be employed to attack a cipher reduced to a high number of rounds.

Then we have applied the rebound attack on the block cipher Camellia where the key size is 128 bits. We have had to modify the technique to attack Camellia since it is not a plain Feistel. We have managed to find a chosen-key distinguisher on the block cipher reduced to 11 rounds with 2^{16} computations. Hash functions are one of the applications of block ciphers. Hence, we have examined the security of the hash functions based on Camellia-128, which are built in Matyas-Meyer-Oseas or Miyaguchi-Preneel modes. As a result, we have used the optimised 3-round inbound phase to generate 4-sum distinguishers on the 7-round hash function and the 9-round compression function, as well as mounting a 9-round half-collision attack. The time complexity of these attacks are respectively, 2^{32}, 2^{40} and 2^{16} computations. Moreover, we have used the proposed 5-round inbound phase to find a collision attack on 9 rounds of the compression function with the computational complexity of 2^{48}.

We have confirmed the results for the chosen-key scheme by computer simulations. Also, by implementing the presented 5-round inbound phase for Camellia-128, we have found many solutions for this phase even with less computations than expected.

REBOUND ATTACK ON FEISTEL NETWORKS AND APPLICATION TO CAMELLIA

5

Preimage Analysis of Grøstl Hash Function

In response to advances in cryptanalysis and new collision attacks to the MD-family of hash functions, the US National Institute for Standards and Technology (NIST) announced a competition aiming at choosing a new stronger cryptographic hash algorithm standard in 2007. The competition called SHA-3 attracted more than 56 submissions (which 52 have advanced to round 1). In December 2010, NIST selected five finalists. They were BLAKE [4], Grøstl [37], JH [108], Keccak [9] and Skein [33]. From the finalists, the Keccak algorithm has emerged as the winner and became the new SHA-3 hash algorithm.

In our cryptanalysis we consider Grøstl, one of the five finalists, and analyse its security. Grøstl is an iterated hash function and its compression function uses two large distinct (fixed-key) AES-like permutations. There are two versions of this algorithm. The original algorithm is referred to as Grøstl-0 [37]. Grøstl [38] is a tweaked version which was proposed to the final round of the competition.

In this chapter, we first explain some related work and preliminary concepts. Then we derive security bounds for the Grøstl compression function and the hash function against (multi-target) preimage attacks in the ideal permutation model and simultaneously correct the previous proof used to obtain bounds for the preimage resistance security. Then considering the Grøstl-0 compression function and using the previous approach to build the internal differential trail between two almost similar looking permutations, we present chosen preimage and multi-target preimage attacks. Consequently, we show chosen-multi-target preimage attacks for up to 8 out of 10 rounds of the compression function and chosen-preimage attack on the 6-round compression function.

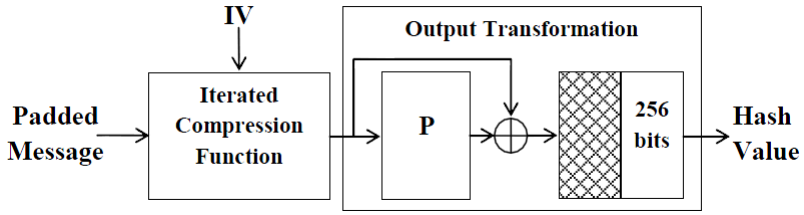
5.1 A Short Description of Grøstl

Here we describe the 256-bit Grøstl-0 hash function [37] which is the main subject of our analysis. Grøstl-0 takes an input message M and splits the padded message into equal length 512-bit blocks $m_1 ... m_t$. The initial value IV is defined as the 512-bit representation of size of the hash value (i.e. 256 bits) which in hexadecimal is 00 00 ... 01 00. Note that IV has only one non-zero byte. Each block is processed by iterating a 512-bit compression function f. At any iteration i, the compression function is defined by

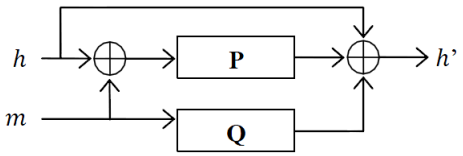
$$f(h_{i-1}, m_i) = P(h_{i-1} \oplus m_i) \oplus Q(m_i) \oplus h_{i-1} = h_i \tag{5.1}$$

where h_{i-1} and h_i are the respective 512-bit input and output chaining values of f and $h_0 = IV$; P and Q are 512-bit 10-round permutations. The 512-bit output value h_t of the final compression function is processed by using an output transformation described by $\text{trunc}_{256}(P(h_t) \oplus h_t)$ where the operation trunc_{256} discards all except the last 256 bits of $P(h_t) \oplus h_t$, to produce a hash value of 256 bits. An abstract view of 256-bit Grøstl-0 is illustrated in Figure 5.1. Often, in our analysis, we denote the pair (h_{i-1}, m_i) with (h, m) and h_i with h'.

The permutations P and Q update an 8×8 state of 64 bytes for 10 rounds. These transformations are similar to that of AES and are described below. For every round r_i, $0 \leq i \leq 9$:



(a) The hash function



(b) The compression function

Figure 5.1: The abstract view of 256-bit Grøstl-0

- *AddRoundConstant (AC)* is the XOR operation of distinct one byte constants to the 8×8 internal states of P and Q. For P, this constant is a round number i which is XORed to the 1^{st} byte of the internal state. For Q, the constant is $i \oplus$ 0xff which is XORed to the 8^{th} byte of the internal state. The permutations P and Q differ only in this step.

- *SubBytes (SB)* is the substitution layer which applies the AES S-box to each byte of the 8×8 state of P and Q.

- *ShiftBytes (SH)* rotates the bytes of the j^{th} row of a state by j positions to the left, where $j = 0, \ldots, 7$.

- *MixBytes (MB)* is the linear diffusion layer in which each column of the state is multiplied with a constant 8×8 circulant MDS matrix.

State S at the round r_i of the compression function is updated by a round transformation defined by $MC_i \circ SH_i \circ SB_i \circ AC_i(S)$ where $i \in \{0, \ldots, 9\}$.

Extensive analysis of Grøstl-0 and its building blocks [39, 45, 72, 73, 74, 85, 94] has motivated the designers to tweak it by making its permutations to look more distinct. Hence, the tweaked hash function is secure against some attacks that apply to Grøstl-0 due to the usage of almost similar permutations. The tweak change the ShiftBytes values for the permutation Q while P is kept the same. The

new left rotation vector is $[1, 3, 5, 7, 0, 2, 4, 6]$, for example the 3$^{\text{rd}}$ raw is rotated to the left by 5 bytes. The other difference between Grøstl and Grøstl-0 is the AddRoundConstant layer. In Grøstl-0 this layer changes different single bytes in P and Q; while in Grøstl it adds different constant to the first row in P and to the last row in Q, also complements the other bytes in Q.

5.2 Related Work

The analytical results on Grøstl-0 mostly have employed the rebound cryptanalysis [72, 73] and its extensions such as Super-Sbox attack [39, 74] and non-full-active Super-Sbox attack [94] to launch collision attacks for the reduced round compression function or hash function [45]. Also the rebound attack is used to find distinguishers for the compression function [39, 57, 85] and permutations [39, 72, 85]. When the permutations are assumed ideal, an l-bit compression function of Grøstl-0 was proved to have a $2^{l/2}$ security against preimage attacks [36, 37] which is also applicable to Grøstl. Grøstl-0 with n-bit hash value has a claimed security of 2^n against preimage attacks [37]. The first analysis of Grøstl with respect to preimage resistance security was done by Wu *et. al.* [109]. They used meet-in-the-middle technique to apply pseudo preimage attack on 5-round Grøstl-256 with the time complexity of $2^{244.85}$ and $2^{230.13}$ memory. Later, pseudo preimage attacks and pseudo 2^{nd}-preimage attacks were applied on up to 6 rounds of the 256-bit Grøstl-0 in [111]. However, it is interesting to see whether the similar looking permutations in Grøstl-0 can be further exploited to mount preimage attacks or their variants on the compression function and the hash function. Later we address this problem by analysing 256-bit Grøstl-0 compression function. Table 5.1 compares our result with the previous preimage attacks applied on 256-bit Grøstl-0.

5.2.1 Super-Sbox Analysis

The Super-Sbox analysis [39, 74] is an improved variant of the rebound technique (was explained in Chapter 4, Section 4.1.1) to solve the inbound phase of two consecutive rounds covering two S-box layers. For Grøstl-0, instead of checking

Table 5.1: Preimage attacks on 256-bit Grøstl-0

Target	Type of Attack	Rounds	Time	Memory	Reference
Hash function	pseudo preimage	5	2^{244}	2^{230}	[109]
Hash function	pseudo preimage	5	2^{239}	2^{240}	[111]
Hash function	pseudo preimage	6	2^{253}	2^{253}	[111]
Hash function	pseudo 2^{nd}-preimage	6	2^{251}	2^{252}	[111]
Compression Function	chosen-2^8-target preimage	6	2^{120}	2^{64}	Sec. 5.4.2
Compression Function	chosen-2^{22}-target preimage	7	2^{120}	2^{64}	Sec. 5.4.2
Compression Function	chosen-2^{136}-target preimage	8	2^{120}	2^{64}	Sec. 5.4.2
Compression Function	chosen preimage	6	2^{128}	2^{64}	Sec. 5.4.3

each S-box for a valid differential as in the basic rebound attack, eight parallel 64-bit S-boxes, called *Super-Sboxes*, are checked. Figure 5.2 illustrates the inbound phase for a sample differential trail of the permutation P or Q in the 256-bit Grøstl-0. In this sample, the inbound phase starts from the state S_1^{SH} before MixBytes of the 1st round and ends at the end of round 3 (i.e the state S_3^{MB}). The first Super-Sbox is highlighted in the figure. In the differential analysis,

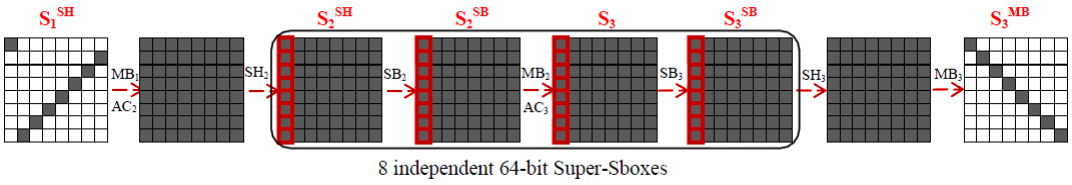


Figure 5.2: Super-Sbox cryptanalysis of the Grøstl-0 permutations

two adjacent ShiftBytes and SubBytes transformations in the first round of the inbound phase can be commuted without changing the final result. Therefore, 8 parallel SubBytes (which is one column of the state), followed by one MixBytes and another 8 parallel SubBytes create one Super-Sbox. AddConstant can be ignored as it does not change the differences. In Figure 5.2, every column at state S_2^{SH} is the input to one 64-bit Super-Sbox and overall there are 8 independent 64-bit Super-Sboxes. Although, the differential distribution table (DDT) for the Super-Sbox includes 2^{128} entries; by fixing the input and output differences of the Super-Sbox, the adversary can check all 2^{64} input values to see if the input difference maps to the output difference.

The inbound phase illustrated in Figure 5.2 is analysed as follows:

1. For all 2^{64} differences at state S_3^{MB}, the adversary computes backward through MixBytes MB_3 and ShiftBytes SH_3 to state S_3^{SB}, then stores the resulting 2^{64} differences for each column in list L_1.

2. He chooses a random difference at state S_1^{SH} and computes the difference forward to state S_2^{SH}. Each column at state S_2^{SH} is the input to one Super-Sbox. For each Super-Sbox at state S_2^{SH}, he connects the input and output differences as follows:

   (a) For the selected Super-Sbox (column) at state S_2^{SH}, he tries all 2^{64} possible pairs of values and calculates the output value of SB_2. Therefore, he finds totally 2^{64} possible differences and corresponding pairs of values for the column at state S_2^{SB}.

   (b) He computes forward to state S_3^{SB} through MixBytes MB_2 and SubBytes SB_3, and finds 2^{64} differences and corresponding pairs of values as the output of the Super-Sbox, then stores them in list L_2.

   (c) The adversary finds a match between differences at list L_1 and corresponding differences and pairs of values at list L_2. Each list has 2^{64} entries and he must match 64 bits (with the probability 2^{-64}). Thus, he finds $2^{64} \times 2^{64} \times 2^{-64} = 2^{64}$ solutions for each Super-Sbox.

3. Since all Super-Sboxes are independent, he finds 2^{64} solutions for each Super-Sbox at the state S_3^{SB} and subsequently for the whole state with the time complexity of 2^{64} computations and 2^{64} memory. Hence, the time complexity of finding each solution is of $O(1)$ on average.

Note that an attacker can still choose 2^{64} differences for the state S_1^{SH} and repeat the inbound phase to find more solutions (pairs of values and differences). In total, he can obtain up to 2^{128} pairs that satisfy the truncated differential path of the inbound phase and they serve as the starting points for the probabilistic outbound phase. The memory complexity for any number of applications of the inbound phase is 2^{64}.

5.2.2 Internal Differential Analysis of Grøstl-0

Peyrin [85] observes that when a cryptographic function is built upon parallel components that are similar then it may be possible to construct a differential trail which spans between these components and such a trail is called the *internal differential trail*. Since the parallel permutations P and Q in the 256-bit Grøstl-0 differ in only one byte of the AddConstant layer, internal differentials can be constructed for the compression function as illustrated in Figure 5.3.

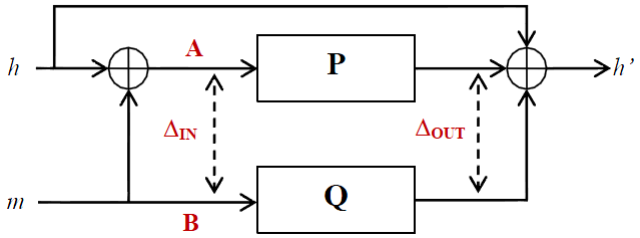


Figure 5.3: The internal differentials for Grøstl-0 compression function

The differences between P and Q of Grøstl-0 compression function can be traced as follows: Let A and B be the input states for P and Q respectively. The input difference of the internal differential trail is given by $\Delta_{IN} = A \oplus B$ and its output difference by $\Delta_{OUT} = P(A) \oplus Q(B)$. Therefore, at any iteration i of the compression function f, we can note that $h = A \oplus B$, $m = B$ and $h' = P(A) \oplus Q(B) \oplus A \oplus B = \Delta_{OUT} \oplus \Delta_{IN}$. The pair $(A, B) = (h \oplus m, m)$ is a valid pair conforming to the internal differential trail. Peyrin combines the internal differentials with the Super-Sbox cryptanalysis to distinguish full Grøstl-0 compression function from the same compression function based on ideal permutations P and Q and mounts a collision attack on the 5-round compression function. The internal differential strategy has been extended by Ideguchi *et al.* [45] to find collisions for the Grøstl-0 hash function reduced to 5 and 6 rounds.

So far, the internal differential analysis has been used to find distinguishers and collision attacks [45, 85]. However, in this chapter we exploit the properties that are weaker than preimage resistance to show chosen-multi-target-preimage attacks on the reduced Grøstl-0 compression function.

5.2.3 Multi-Target Preimage Attack

In a preimage attack on an n-bit hash function H, given a hash value y, an adversary should find a message M such that $H(M) = y$ in less than 2^n evaluations of H. Similarly, in a preimage attack on a l-bit compression function f, given an output y, the adversary should find an input chaining value and message block pair (h, m) such that $f(h, m) = y$. The idea of multi-target (aka one-of-many) preimage attacks on hash functions was first proposed by Merkle [78]. In this attack on an n-bit hash function H, the attacker aims to find a preimage for one of the K specified hash values in less than $2^n/K$ evaluations of H [78, 88]. Similarly, on a l-bit compression function f the task is to find a pair (h, m) which hits one of K specified outputs of f in less than $2^l/K$ evaluations of f.

An interesting application of the multi-target preimage attack is when an attacker has access to a password file containing user names and the corresponding hashes of the passwords, so he can find the password of one of the users by attacking these hash values. As noted in [87], security against these attacks is also necessary in applications that combine the hash trees [76] and digital time-stamps such as in the RFC standard of Evidence Record Syntax [100]. The multi-target preimage attack is used in the cryptanalysis of hash functions to find second preimages [2] for some popular hash function construction modes such as Merkle-Damgård. It is also used to find preimages for the reduced round SHA-256 and SHA-512 [41].

In a slightly different setting, an attacker may show the existence of a set of outputs for either a hash function H or a compression function f where a preimage can be found for one of the outputs in the set. We call attacks in this setting chosen-multi-target preimage attacks or chosen-K-target preimage attacks for an output set of size K.

5.3 The Generic Security of Grøstl Against Multi-Target Preimage Attacks

5.3.1 Security Bounds for the Compression Function

It has been proved in [36, Proposition 2], that for a l-bit compression function with two parallel permutations (such as Grøstl), an adversary find a preimage with the advantage of at most $q^2/2^l$ when he makes at most q queries to these permutations and their inverses. Hence, the complexity of a preimage attack on Grøstl compression function is about $2^{l/2}$ which is also the claimed security bound in [37].

We remark that the security proof in [36] which led to the above result is not completely accurate. Because it assumes for an input query to a permutation, the output is chosen uniformly random from the space of $\{0,1\}^l$. This proof does not take into consideration that P and Q are permutations and the queries are made to the same permutations. Assume, P has been queried i times by inputs $\alpha_1, \ldots, \alpha_i$, returning values $\beta_1 = P(\alpha_1), \ldots, \beta_i = P(\alpha_i)$, the response for the next query α is not among the previous responses $\beta_1, \ldots, \beta_i$. Thus, the returned value $\beta = P(\alpha)$ is uniformly random over $2^l - i$ values (i.e all values except $\beta_1, \ldots, \beta_i$).

Considering this improvement we extend the analysis to the security of the construction against K-target preimage attacks. In Theorem 5.1, we prove that for an adversary who makes q queries to the permutations, the success probability of a preimage attack on l-bit Grøstl compression function is upper bounded by $\frac{(q^2+q)}{2^l}$, where $q^2 + q \leq 2^l$ (under the assumption that the permutations P and Q are ideal). This proof is more accurate compared to the one in [36]. Also, Theorem 5.1 shows that the success probability of a K-target preimage attack on Grøstl compression function after q queries is upper bounded by $K \cdot \frac{(q^2+q)}{2^l}$, where $q^2 + q \leq 2^l$. Therefore, an adversary must make at least $2^{l/2}/\sqrt{K}$ queries to find a preimage of one of the K targets.

Theorem 5.1. *Consider an l-bit Grøstl compression function based on ideal permutations P and Q. For a computationally unbounded adversary $\mathcal{A}$ making at most q queries to the permutations P and Q and their inverses, the advantage to find a preimage for Grøstl compression function is upper bounded by $\frac{(q^2+q)}{2^l}$ where*

$q^2 + q \leq 2^l$. *When* $\mathcal{A}$ *is supplied with* K*-target hash values, the advantage to find a preimage for one of these hash values is upper bounded by* $K \cdot \frac{(q^2+q)}{2^l}$ *where* $q^2 + q \leq 2^l$.

Proof. Let x be the value which we desire to find its preimage for the Grøstl compression function. Hence, we need to find α, α' (i.e. inputs of P and Q, respectively) such that $x \oplus P(\alpha) \oplus \alpha \oplus Q(\alpha') \oplus \alpha' = 0$. Suppose that $\mathcal{A}$ has made two sets of queries as follows:

1. A total of i_1 queries to P, P^{-1} oracles and stored the input/output pairs $(\alpha_j, \beta_j = P(\alpha_j)), 1 \leq j \leq i_1$ for these i_1 queries in list L_1.
2. A total of i_2 queries to Q, Q^{-1} oracles and stored the input/output pairs $(\alpha'_j, \beta'_j = Q(\alpha'_j)), 1 \leq j \leq i_2$ for these i_2 queries in list L_2.

Now we claim that the probability that the next query of $\mathcal{A}$ results in a preimage is upper bounded by

$$p_{next} \leq \begin{cases} i_2/(2^l - i_1) & \text{if the next query is to } P \text{ or } P^{-1} \\ i_1/(2^l - i_2) & \text{if the next query is to } Q \text{ or } Q^{-1} \end{cases}$$

The justification for the first bound is as follows: Suppose that $\mathcal{A}$'s next query is to P by the value α, where α is not equal to one of the α_j's already in the list L_1. The returned value $\beta = P(\alpha)$ is chosen uniformly random from the set of all $2^l - i_1$ (l-bit strings which are not equal to one of the β_j values in L_1). In order to find a preimage, β must be such that for a $j \in \{1, ..., i_2\}$,

$$x \oplus \beta \oplus \alpha \oplus Q(\alpha'_j) \oplus \alpha'_j = 0 \tag{5.2}$$

However, for each value of j, there is at most one value of β such that Equation (5.2) is satisfied. Therefore, there are at most i_2 "good" values of β that produce a preimage, and since β is chosen uniformly among $2^l - i_1$ possible values, the probability of β being "good" is at most $i_2/(2^l - i_1)$ (as in the bound above). A similar argument applies to P^{-1} queries and a symmetric argument applies for queries made to Q or Q^{-1} giving the bound $i_1/(2^l - i_2)$.

We conclude that if $\mathcal{A}$ has been made totally $i_1 + i_2 = q - 1$ queries to all oracles, then the next q^{th} query provides a preimage with probability $p_q \leq q/(2^l - q)$, since

both bounds for p_{next} above are below this value. Therefore, the probability of a preimage being found in any of q queries is at most

$$p_{preimage} \leq \sum_{i=1,\ldots,q} p_q \leq \sum_{j=1,\ldots,q} j/(2^l - j) \leq q^2/(2^l - q) \leq q^2 + q/2^l \tag{5.3}$$

iff $q^2 + q \leq 2^l$. This result is very close to the one claimed in [36].

Now assume that $\mathcal{A}$ is challenged with $K = 2^{s_K}$ target hash values, of which it needs to find a preimage for one of them. Equivalently, $\mathcal{A}$ has to find a preimage to a subset of $(l - s_K)$ output bits of the compression function. Therefore, Equation (5.2) can be modified to only look at the equality on these $l - s_K$ output bits. Let l-bit state of Grøstl is viewed as a byte oriented $b \times b$ matrix, that is $l = b \times b \times 8$ bits with matrix indices $1, \ldots c$ (where $c = b \times b$). Now Equation (5.2) can be modified as follows: Let Z denote the subset of $\{1, \ldots, c\}$ corresponding to the indices of $(l - s_K)/8$ bytes in the $b \times b$ matrix (i.e. the size of Z is $(l - s_K)/8$ bytes). For an $b \times b$ byte variable x, let $(x)_Z$ denote the restriction of x to the bytes in indices contained in Z. Then the modified Equation (5.2) is:

$$(x)_Z \oplus (\beta)_Z \oplus (\alpha)_Z \oplus (Q(\alpha'_j))_Z \oplus (\alpha'_j)_Z = 0 \tag{5.4}$$

This equation determines $(\beta)_Z$ uniquely, once x, α, α'_k and $Q(\alpha'_k)$ are determined. Since the values of α and β are still l bits long, for each value of $j = 1, ..., i_2$, there are at most $K = 2^{s_K}$ "good" values of β that satisfy Equation (5.4). Totally, there are at most $i_2 \cdot K$ "good" βs for all values of j. Hence, the bounds for p_{next}, and the final result in Equation (5.3) gets multiplied by K. Therefore, the advantage of $\mathcal{A}$ to find a preimage for one among $K = 2^{s_K}$ target hash values is at most $K.(q^2 + q)/2^l$. □

The above analysis is also applicable for Grøstl-0 as permutations are assumed ideal, and the security bound derived for Grøstl compression function in Theorem 5.1 does not depend on the details of P and Q permutations. Note that, it is reasonable to use the above bound (i.e. $2^{l/2}/\sqrt{K}$) for the generic security of the construction against chosen-K-target preimage attacks.

5.3.2 Security Bounds for the Hash Function

Suppose a chosen-K-target preimage attack on the $2n$-bit Grøstl compression function extends to a K^*-target preimage attack on the n-bit hash function. We prove in Theorem 5.2 that a preimage for one of K^* hash values of Grøstl hash function is found with a probability less than $\frac{q.2^n.K}{2^{2n}-(K+s-1)} + \frac{(q^2+q).K}{2^{2n}}$, where q is the number of queries and $q^2+q \leq 2^{2n}$. The probability $\frac{q.2^n.K}{2^{2n}-(K+s-1)}$ is approximately upper bounded by $q.K/2^n$ when $K+q$ is much smaller than 2^{2n}. Therefore, the total success probability is less than $\frac{q.K}{2^n} + \frac{(q^2+q).K}{2^{2n}}$. The first term dominates the second term, so the adversary has to make at least $2^n/K$ queries to find a preimage of one of the K^* target hash values. Note that permutations are assumed ideal and the analysis is independent of the internal permutations of P and Q, so this analysis is also valid for the Grøstl-0 hash function.

Theorem 5.2. *Consider an n-bit Grøstl hash function with the compression function f based on the ideal permutations P and Q, and output transformation OT as $OT(x)=\text{trunc}_n(P(x)\oplus x)$. Let Y be a set of the compression function outputs which are selected independently of P and Q, and Y^* is the target hash value set such that $Y^* = \{OT(x) : x \in Y\}$. The probability of finding a preimage of one of the elements of Y^* is upper bounded by $\frac{q.2^n.K}{2^{2n}-(K+s-1)} + \frac{(q^2+q).K}{2^{2n}}$, where $q^2+q \leq 2^{2n}$ (Note that K^* and K are respectively the number of elements of the sets Y^* and Y).*

Proof. Let x be the preimage for one of the hash targets in set Y^*. We can split the results into two independent cases as follows:

- *Case I*: x is the preimage of one of the targets in set Y^* such that $f(x) \in Y$.
- *Case II*: x is the preimage of one of the targets in set Y^* such that $f(x) \notin Y$.

The success probability of finding x as the preimage of one of the targets in Y^* is $P(\text{case I}) + P(\text{case II})$. We know from Theorem 5.1 that $P(\text{case I}) \leq (q^2+q)\cdot K/2^{2n}$. Assume there exists a query number s to P or to P^{-1} that $(z_s^*, P(z_s^*))$ denotes the input and output of P for the s^{th} query, and $z_1, ..., z_K$ denote the elements of Y. From case II we get $i \in \{1, ..., K\}$ such that:

$$z_s^* \notin Y \text{ and } \; trunc_n(P(z_s^*) \oplus z_s^*) = trunc_n(P(z_i) \oplus z_i) \tag{5.5}$$

Let us consider the case that the s^{th} query is a query to P (Note that the case when it is a query to P^{-1} is similar). Given any fixed values for z_s^*, z_i and $P(z_i)$, there are 2^n values of $P(z_s^*) \in \{0,1\}^{2n}$ that satisfy Equation (5.5). Since there are K possible values for i, there are at most $2^n \cdot K$ "bad" values of $P(z_s^*)$ that satisfy Equation (5.5) for some i. We call the set of "bad" $P(z_s^*)$ values Bad_s. Now, if the values of $P(z_r)$, $r = 1, ..., K$, and $P(z_r^*)$, $r = 1, ..., s-1$, are already fixed, the value of $P(z_s^*)$ is uniformly random in a set of size $2^{2n} - (K + s - 1)$, which is returned as the answer to the adversary's s^{th} query z_s^*. Therefore, the probability that $P(z_s^*) \in Bad_s$ is at most $p_s = \frac{|Bad_s|}{2^{2n}-(K+s-1)} \leq \frac{2^n \cdot K}{2^{2n}-(K+s-1)}$. Taking a union over all possible $s = 1, ..., q$, we obtain that $P(\text{case II}) \leq \sum_{s=1,...,q} p_s \leq \frac{q \cdot 2^n \cdot K}{2^{2n}-(K+q)}$. Hence, the success probability of finding preimage of one of the elements of Y^* is (i.e. $P(\text{case I}) + P(\text{case II})$) is upper bounded by $\frac{(q^2+q)\cdot K}{2^{2n}} + \frac{q.2^n \cdot K}{2^{2n}-(K+s-1)}$, where $q^2 + q \leq 2^{2n}$. □

5.4 Attacks on Grøstl-0 Compression Function

In this section, we analyse reduced round variants of the 256-bit Grøstl-0 compression function against chosen-multi-target preimage attacks. In our attacks on the considered design variants, we show the existence of sets of K-target outputs where it is possible to find a preimage for at least one of the targets in each of these sets. The complexity of our attacks are less than the generic attack complexity obtained in Section 5.3.1, when the permutations are assumed ideal. We apply the idea of internal differences between similar looking permutations to build internal differential trails and find chosen-multi-target preimages. These trails are built by using Super-Sbox cryptanalytical technique. We can find chosen-multi-target preimages for up to 8 rounds of the compression function. In some cases, chosen-multi-target preimage attacks can be converted to chosen-preimage attacks on the compression function. Consequently, we show a chosen-preimage attack on 6-round compression function. By controlling the initial state of the compression function and forcing it to be the initial value (IV) of the hash function, the chosen-preimage attack is found for up to 5 rounds of the compression function. Our results are summarized in Table 5.2, note that the memory complexity for all the attacks is 2^{64}.

Table 5.2: Summary of the results on 256-bit Grøstl-0 compression function

Attack	Targets	Rounds	Time complexity	Generic complexity	Section Section
Chosen-multi-target preimage	2^8	6	2^{120}	2^{252}	5.4.2
	2^{64}	6	2^{64}	2^{224}	5.4.2
	2^{22}	7	2^{120}	2^{245}	5.4.2
	2^{78}	7	2^{64}	2^{217}	5.4.2
	2^{136}	8	2^{120}	2^{188}	5.4.2
	2^{192}	8	2^{64}	2^{160}	5.4.2
Chosen preimage	1	5 ($h = IV$)	2^{144}	2^{256}	5.4.3
	1	6	2^{128}	2^{256}	5.4.3

5.4.1 Abstract View of the Attacks

In our attacks, we build internal differential trails between the permutations P and Q using Super-Sbox technique, such that the complexity of a chosen-K-target preimage attack is less than $2^{l/2}/\sqrt{K}$. An internal difference is the difference between P and Q. The size of K is influenced by the factors such as the number of attackable rounds, the number of solutions obtained from the inbound phase of the Super-Sbox analysis and the amount of control we would like to exert at the input state of the compression function. Recall from Section 5.2.2, if A and B are inputs to the permutations P and Q, the compression function following the internal differential trail can be described as $f(h, m) = h' = \Delta_{IN} \oplus \Delta_{OUT}$ where $\Delta_{IN} = A \oplus B$ and $\Delta_{OUT} = P(A) \oplus P(B)$.

Once the internal differential trail is built for a possible number of rounds, we take all possible outputs of this trail as the K-chosen-target set. Then we can find a preimage (h, m) for one of the K targets as follows. Since $A \oplus B = \Delta_{IN} = h$, when we extend the difference between the pair of values that satisfy the inbound phase in the backward trail of the outbound phase, we eventually end up with a pair of values (A, B) with the difference h. Note that the message block $m = B$ (i.e. the input value to permutation Q). By computing the forward trail of the outbound phase with the solution which satisfies both the inbound and outbound phases, we end up hitting at least one of the K target values. A matched target

value is defined by $h' = \Delta_{IN} \oplus \Delta_{OUT}$. Thus, we have found the pair $(h, m) = (\Delta_{IN}, B)$ which hits one of the chosen-K-target values. If more than one solution satisfies the internal differential trail then we can find preimages for more than one K target outputs.

Note that the target value set is independent of P, Q and the input values. Therefore these attacks are distinct from the trivial scenarios where one can compute a set of hash values for some arbitrary messages and later show a preimage for one of the hash values.

5.4.2 Chosen-Multi-Target Preimage Attacks

Attacks on 6-Round Compression Function

Figure 5.4 shows the truncated internal differential trail for 6-round compression function where A and B are the inputs to the permutations P and Q and their difference $A \oplus B$ is the input chaining value h for the compression function.

In our trails, we use the labels *controlled* and *uncontrolled* values to refer to the actual values of the states. Controlled values are known to the adversary from the beginning of the attack. For example in Figure 5.4, at the state h' there are 56 controlled bytes and their values are fixed to zero. On the other hand, values of uncontrolled bytes are not known until the completion of the attack. Assume set K is the target set which has 2^{64} 512-bit values. Every element of K consists of only 8 uncontrolled bytes at the positions shown in the state h'. The application of Super-Sbox attack on these 6 rounds finds the preimage (h, m) of the compression function for a $h' \in K$.

The attack works as follows:

- *The inbound phase* starts at the state after SH_1 and ends before SB_4 (it is illustrated by the dashed lines in Figure 5.4). This phase is solved by the application of the Super-Sbox analysis, and provides up to 2^{128} starting points for the outbound phase. The complexity of this phase is 2^{64} precomputations and 2^{64} memory.

- *The outbound phase*

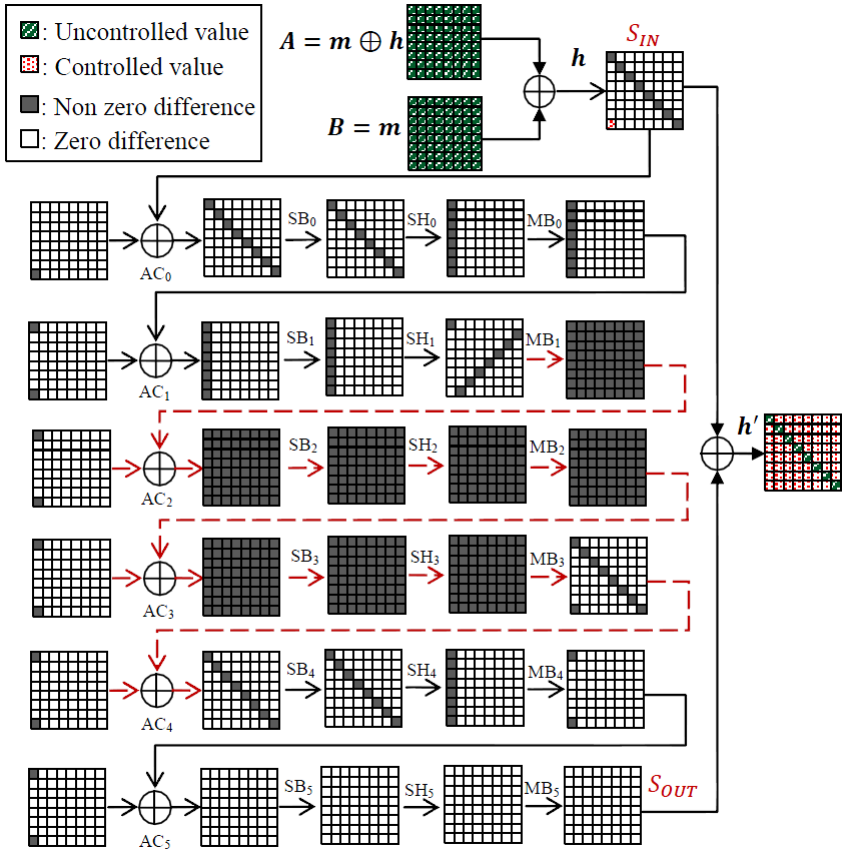


Figure 5.4: 6-round internal differential trail with 2^{64} targets

1. Use the starting points of the inbound phase to compute forward from the state before SB_4 to S_{OUT}. This part is satisfied with the probability of 2^{-64}. Because, the $8 \mapsto 2$ bytes difference propagation through MB_4 is successful with the probability of 2^{-48} and two bytes differences are cancelled by AC_5 with the probability of 2^{-16}.

2. Use the corresponding starting point of the pair satisfying Step 1 and work backward from the state before MB_1 to the input state S_{IN}. This step has a success probability of one.

3. The XOR addition of the input difference h at state S_{IN} and the output difference at S_{OUT} is h', which is one of the 2^{64} target output values of the compression function.

4. The pair of values that obtains the difference at S_{IN} are the inputs of P and Q, their difference is the input chaining value h and the input value of Q is the message block m. Thus, we found the pair (h, m) for an output chaining value h' which is one of the target outputs.

The time complexity of the attack is 2^{64} computations due to the outbound phase and requires 2^{64} memory for the inbound phase. This is much less than the generic complexity of 2^{224} to find a preimage for one among 2^{64} target compression function outputs.

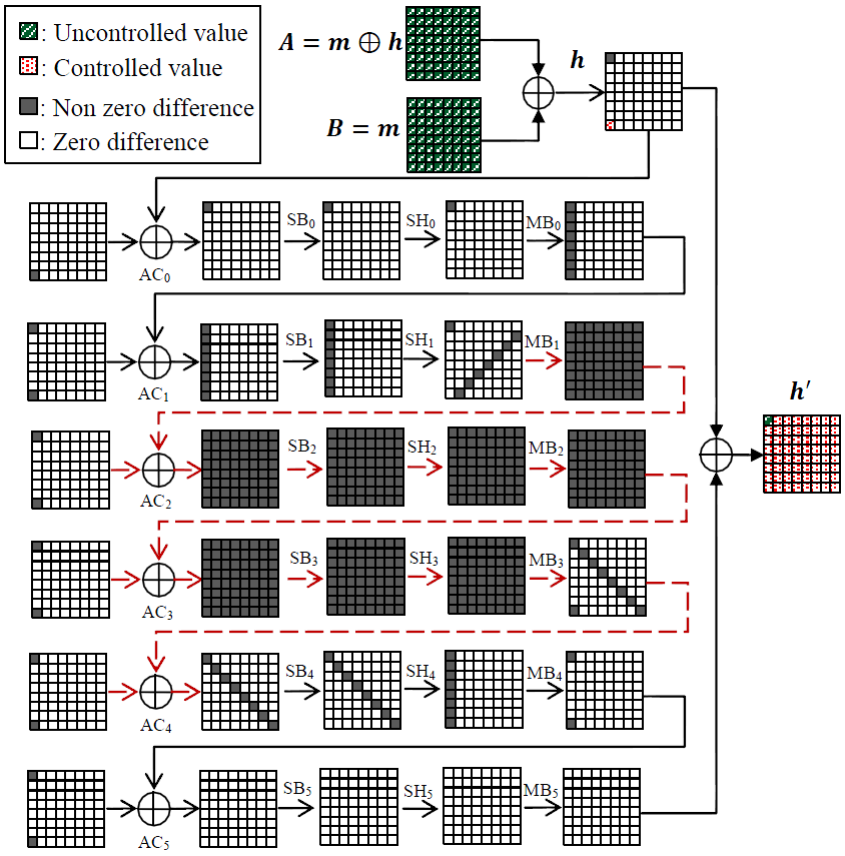


Figure 5.5: 6-round internal differential trail with 2^8 targets

Another 6-round internal differential trail is illustrated in Figure 5.5 where it has 2^8 targets. Similar to the previous 6-round attack, the inbound phase has at most 2^{128} solutions as the starting points of the outbound phase. The inbound phase needs 2^{64} states of memory. The outbound phase is successful

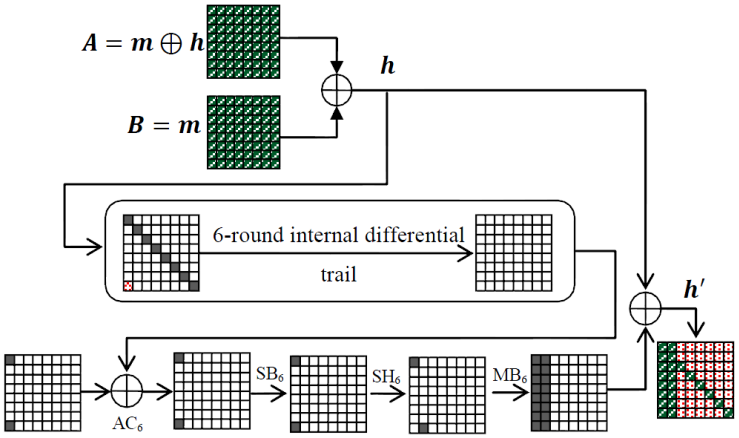
with the probability of 2^{-120}. The probabilistic transformations of the outbound phase are, the $8 \mapsto 2$ difference propagation through MB_4 (i.e. 2^{-48}), two bytes difference cancellation by AC_5 (i.e. 2^{-16}), and the $8 \mapsto 1$ difference propagation through the inverse MB_0 (i.e. 2^{-56}). So, the complexity of the chosen-2^8-target preimage attack is 2^{120} time and 2^{64} memory. The generic complexity of a 2^8-target preimage attack is 2^{252}.

Attacks on 7 and 8 Rounds of the Compression Function

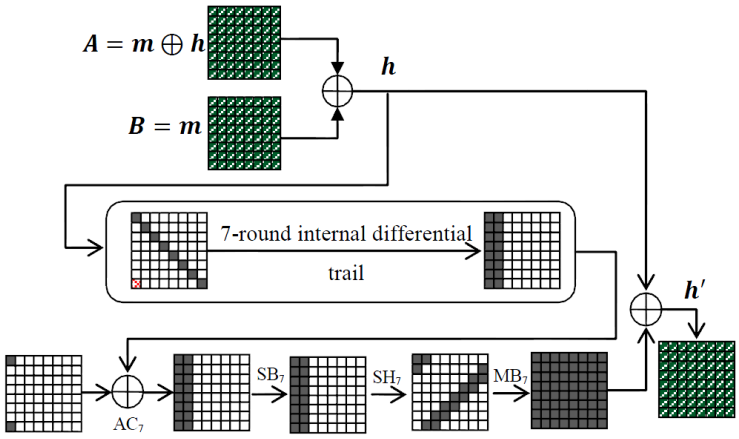
The internal differential trail of Figure 5.4 can be extended to 7 and 8 rounds as shown in Figure 5.6.

Figure 5.6a illustrates the addition of one round to the 6-round trail of Figure 5.4. The 7-round trail is used for the 2^{78}-target preimage attack on the 7-round compression function with no extra cost, as the internal differentials of 7^{th} round hold with the probability of one. The input differences coming from the final round constants are known and the S-boxes are the AES sbox. Therefore, for each active input difference we get 127 possible output differences through SB_6. Totally, there are about 2^{14} possible differences going to the MixBytes in the final round. MB_6 is a linear transformation which maps 2-byte differences to 16-byte differences. Hence, for all 2^{14} possible differences in the state before MB_6, 2^{14} differences are generated for the state after MB_6. Since for every value among 2^{64} possibilities in h, there are 2^{14} values in the final state, there are up to 2^{78} possibilities for the state h'. This attack finds the chosen-2^{78}-target preimage with 2^{64} time complexity and 2^{64} memory, while the generic attack needs 2^{217} computations.

The 7-round trail of Figure 5.6a is extended by one more round and obtains an 8-round trail as shown in Figure 5.6b. Again the 8^{th} round probability is one. Here MB_7 propagates the state of 16 non-zero differences to a full active state. However, due to the linearity of MB_7, there are 2^{128} possible values for the full active state. Since, there are 2^{64} possibilities for h, there are totally 2^{192} possible values for h' and hence 2^{192} targets for the compression function. The complexity of the chosen-2^{192}-target preimage is 2^{64} time and 2^{64} memory. The generic complexity of this attack is 2^{160} computations.



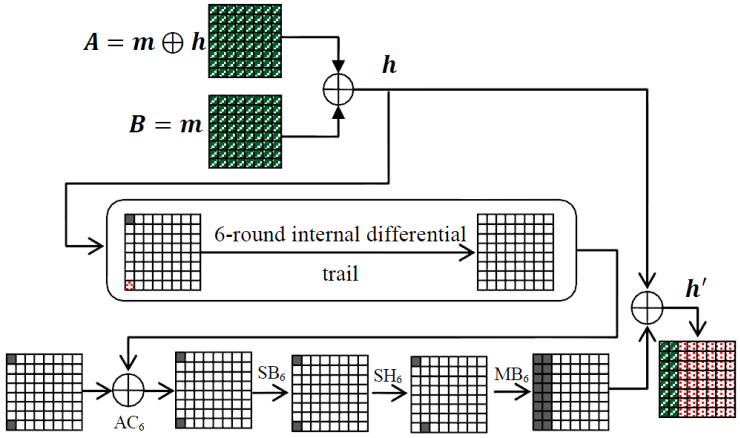
(a) 7-round internal differential trail for 2^{78} targets



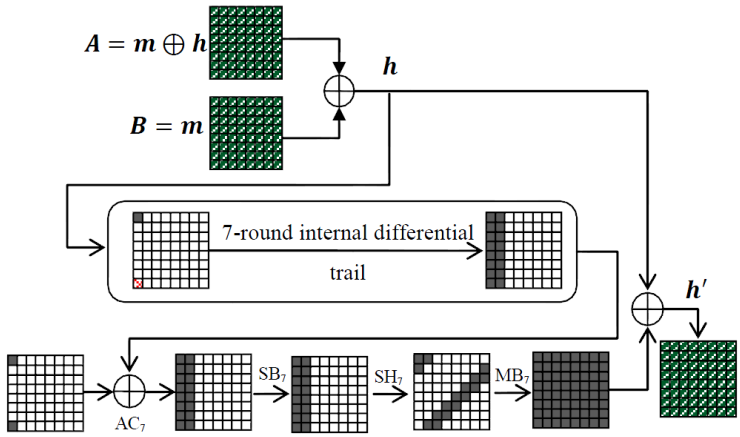
(b) 8-round internal differential trail for 2^{192} targets

Figure 5.6: Extension of the internal differential trail of Figure 5.4

Similarly, the 6-round internal differential trail of Figure 5.5 is also extended to 7 (see Figure 5.7a) and 8-round (see Figure 5.7b) internal differential trails to apply respectively 2^{22} and 2^{136}-target preimage attacks on the 256-bit Grøstl-0 compression function. Note that both extensions follow the outbound phase of 6-round internal differential trail of Figure 5.5 with probability one. Therefore, the complexity of the 2^{22}-target preimage attack on 7-round compression function as well as the 2^{136}-target preimage attack on 8 rounds are 2^{120} time and 2^{64} memory.



(a) 7-round internal differential trail for 2^{22} targets



(b) 8-round internal differential trail for 2^{136} targets

Figure 5.7: Extension of the internal differential trail of Figure 5.5

5.4.3 Chosen Preimage Attacks

In the preimage attack on a compression function f, the attacker who knows the specific output h' of f aims to find a pair (h, m) such that $f(h, m) = h'$. It is possible to convert a chosen-multi-target preimage attack on a compression function to the chosen-preimage attack if there is sufficient degrees of freedom available from the inbound phase of the attack. We can achieve this by using the freedom to repeat the inbound phase and therefore the multi-target attack itself until we hit all the compression function output targets.

Chosen Preimage Attacks on 6 Rounds

We convert the 2^{64}- and 2^{8}-target preimage attacks on the 6-round compression function described above (by Figures 5.4 and 5.5, respectively) to chosen-preimage attacks. By repeating 2^{8} times the former and 2^{64} times the latter we aim to find preimages for the whole elements of the target sets. In both attacks the overall complexity is 2^{128} time and 2^{64} memory.

Chosen Preimage Attacks on 5 Rounds Where $h = IV$

Figure 5.8 illustrates a 5-round internal differential trail which exerts control over the input chaining value h and makes it equal to IV of 256-bit Grøstl-0. There are 2^{64} targets for this attack. Note that these targets have a fixed non-zero byte same as the only non-zero byte of IV of 256-bit Grøstl-0 at the same position.

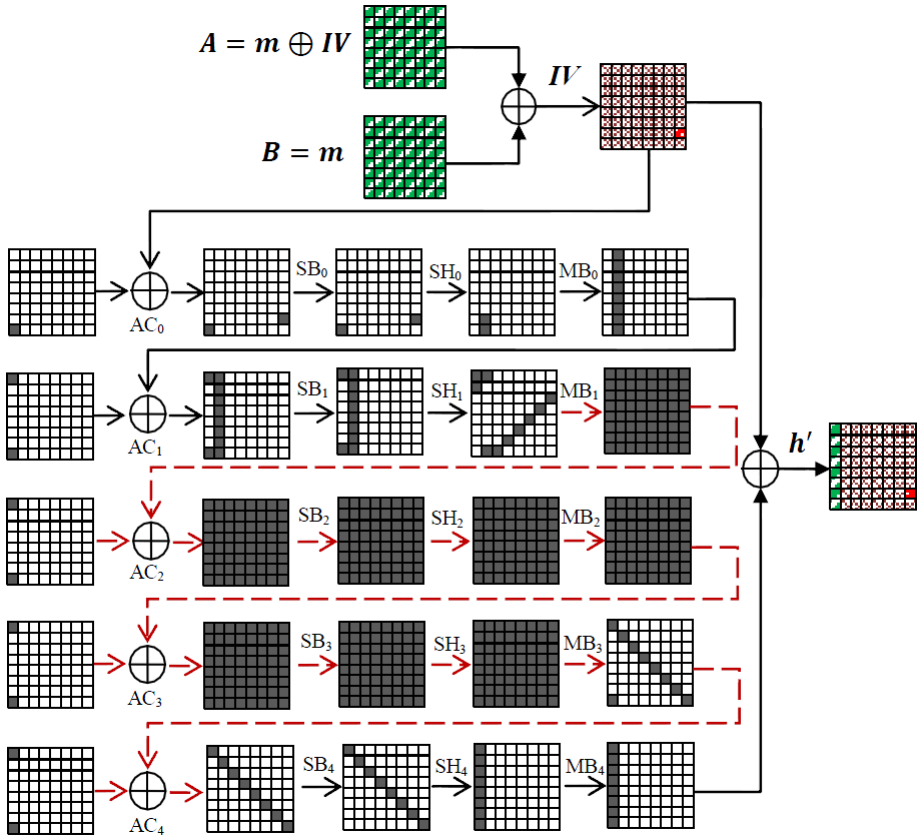


Figure 5.8: 5-round internal differential trail with 2^{64} targets

The inbound phase of the attack starts at the 10-byte active state before MB_1 and ends at the 8-byte active state before SB_4. For a difference chosen at the state before MB_1 the inbound phase results in 2^{64} pairs of values with 2^{64} time and 2^{64} memory complexity. Overall, the inbound phase can produce up to $2^{64+80} = 2^{144}$ pairs of values as the starting points for the outbound phase.

The forward trail of the outbound phase holds with the probability one. In the backward trail, at first the inverse AC_1 cancels 2 active bytes in the first column of the state before SB_1 with the probability of 2^{-16}. The $8 \mapsto 2$ active byte propagation through the inverse MB_0 has the success probability of 2^{-48}. Finally, with 2^{-8} probability, the active byte in the first column before SB_0 is cancelled and with the probability of 2^{-8} the active byte of the input state S_{IN} can be forced to be equal to the non-zero byte in IV. Overall, the success probability of the outbound phase is 2^{-80}. Therefore, an adversary finds one of the 2^{64} targets with the total complexity of 2^{80} computations and 2^{64} memory. By repeating this part 2^{64} times, the adversary expects to find the preimage of all the 2^{64} targets. The time complexity of the attack is 2^{144} while it needs 2^{64} memory. Note that the inbound phase of the internal differential trail of Figure 5.8 starts with 10 active bytes and ends with 8 active bytes on the other side. Using the method described in Section 5.2.1 there are 2^{144} solutions for the inbound phase. Thus, there are enough freedom to find all the preimages.

5.5 Conclusion

Grøstl-0 was one of the hash algorithms submitted to the NIST SHA-3 competition and selected as a finalist. In the final round of the competition, the design was tweaked (i.e. called Grøstl from then) to make two almost identical underlying permutations behave differently. In this chapter, we have corrected the previous security proof for the preimage resistance of the compression function and proved that the generic complexity of K-target preimage attack on the l-bit Grøstl compression function is at least $2^{l/2}/\sqrt{K}$. Also we have proved that a chosen-K-target preimage attack on the Grøstl-0 compression function is extended to a K^*-target preimage attack on the hash function with at least $2^n/K$ queries.

Then we have employed the internal differential trails combined by the super-sbox analysis on the compression function of Grøstl-0 to present chosen-multi-target preimage attacks for up to 8 rounds, and chosen-preimage attacks on 5 and 6 rounds of the compression function. These attacks do not apply to the compression function of Grøstl as distinct permutations in Grøstl completely prevent the application of internal differentials even for few rounds.

6 Conclusion

Design and analysis of symmetric algorithms, such as block ciphers and hash functions, have recently attracted many cryptographic researches. In this thesis we have explored different types of block ciphers and hash functions and their analytical methods. Consequently, we have been able to present new techniques for the cryptanalysis of well-known algorithms including LBlock, Camellia and Grøstl. Our results are summarised bellow.

Truncated Differential Analysis

The truncated differential analysis presented in this thesis concentrates on differential probability distribution rather than on a small collection of fixed differences. This analysis can be applied to the cryptanalysis of symmetric algorithms and requires less data to launch an attack than the classical differential attack. We have presented a new framework for the analysis of LBlock lightweight block cipher using the truncated differential analysis. Taking advantage of the LBlock key schedule, we are able to extend the attack and combine it with the fixed

differential at the beginning of the truncated differential trail. We have used the LLR statistical test and KL-divergence as the distinguishing tools. Computer implementation and experiments confirm our theoretical results and observations. Finally, we have applied the key-recovery attack on LBlock reduced to 18 rounds and the related-key attack on 21 rounds, where the former requires only 2^{23} and the latter 2^{30} plaintext/ciphertext pairs to succeed. The efficiency of these attacks is better than the generic exhaustive search attacks.

Rebound Attack on Feistel Networks

The known-key and chosen-key attacks analyse the security of block ciphers where the adversary knows the key or can choose it himself. These attacks are important when the cipher is used to build a hash function. We have reviewed the known-key (and chosen-key) attacks on Feistel-SP networks by the rebound attack, and improved the previous results. Using the proposed method, we have analysed Camellia block cipher. A chosen-key distinguisher has been presented for Camellia-128 reduced to 11 rounds with 2^{16} computations complexity. We have implemented the attack and the experiments confirm the theory. The rebound attack has been optimised to attack the hashing modes of Camellia-128. The hash function is built based on the Matyas-Meyer-Oseas or Miyaguchi-Preneel schemes. Thus, we apply collision attacks and 4-sum distinguishers on up to 9 rounds of the compression function.

Grøstl Preimage Analysis

Grøstl was a finalist in the NIST SHA-3 hash function competition. Grøstl is a tweaked variant of its predecessor Grøstl-0, a second round SHA-3 candidate which was selected as the finalist. We have corrected the security proof for the preimage resistance of any compression function which is built based on two parallel permutations, such as Grøstl. Accordingly, the security bounds against the multi-target preimage attacks have been proven for the Grøstl compression function and the hash function. These security bounds are applicable to any Grøstl-like design. Then, we have analysed the security of the Grøstl compression function, reduced to up to 8 rounds, against multi-target preimage attacks for

chosen target sets. We have used internal differential and Super-Sbox analysis techniques to mount the attacks. Consequently, it is possible to find preimage attacks on up to 6 rounds of the compression function for specified sets of output values.

Future Work

The truncated differential distribution analysis is a new type of differential cryptanalysis. Using the probability distribution of the differences for individual state symbols is the advantage of this technique over the other differential distribution analyses which find the probability distribution for the whole state. Therefore, the truncated differential distribution analysis is applicable to block ciphers with relatively large states. However, the way the differential distribution table is built is a challenge in ciphers which imply dependencies between the symbols. A direction for future research could be computing the differential distribution table taking into account the dependencies. Also, the way we extended the truncated differential analysis, by taking the advantage of the LBlock key schedule, can be used for the analysis of other designs. Moreover, it is possible to combine the probability distribution of differences with other statistical attacks such as the linear cryptanalysis and rotational analysis. For example, the differential probability distribution is applied together with linear cryptanalysis in [17].

The rebound attack is a very popular method for the analysis of hash functions. The attacks we presented for the analysis of Camellia and Grøstl can be applied to other hash functions and block ciphers. Furthermore, the multi-target preimage attacks on the 256-bit Grøstl can be easily extended to the 512-bit variant. It may be possible to reformulate the rebound attack on the Feistel-SP networks, so that it can be applied to the other Feistel networks where the round function is of a different kind, such as ARX.

A

Camellia-128 Key Schedule

The key schedule of Camellia is explained in [3] for all the key sizes. In Camellia-128, the 128-bit master key K is used to generate 18 round sub-keys, and 8 supplementary sub-keys for the whitening and FL layers. The key K is encrypted by 2 rounds of Camellia and then get mixed with itself. Then the result is again encrypted by another 2 rounds. The output is named K_A. Figure A.1 describes the procedure of producing K_A from K. The constant sub-keys $\Sigma_i, 1 \leq i \leq 4$ are used for the encryption, as illustrated in Table A.1.

Table A.1: The key-schedule constant sub-keys

$Sigma_1$	`0xA09E667F3BCC908B`
$Sigma_2$	`0xB67AE8584CAA73B2`
$Sigma_3$	`0xC6EF372FE94F82BE`
$Sigma_4$	`0x54FF53A5F1D36F1C`

The 64-bit round sub-keys $k_1, k_2, \ldots, k_{18}$, whitening keys $kw_1, \ldots, kw_4$, and FL-layer sub-keys $kl_1, \ldots, kl_4$ are all generated by rotating K and K_A and taking the left or the right half of them. Table A.2 shows how the sub-keys are generated.

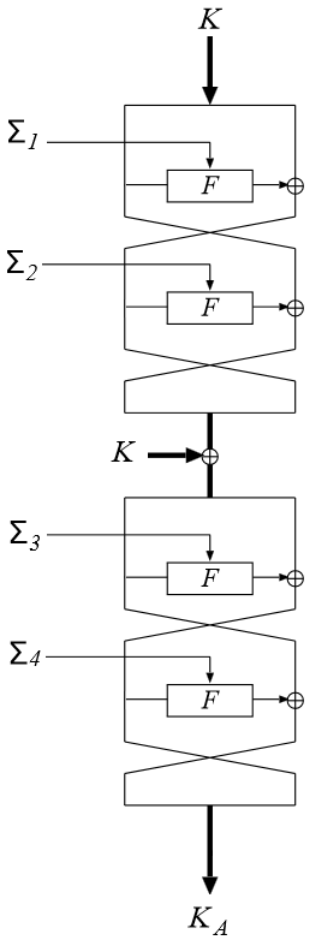


Figure A.1: Camellia-128 key schedule [3]

Table A.2: Sub-keys for Camellia-128

Type	Sub-key	Value
Whitening key	kw_1	$(K \lll 0)_L$
	kw_2	$(K \lll 0)_R$
Round key	k_1	$(K_A \lll 0)_L$
	k_2	$(K_A \lll 0)_R$
	k_3	$(K \lll 15)_L$
	k_4	$(K \lll 15)_R$
	k_5	$(K_A \lll 15)_L$
	k_6	$(K_A \lll 15)_R$
FL FL^{-1}	kl_1	$(K_A \lll 30)_L$
	kl_2	$(K_A \lll 30)_R$
Round key	k_7	$(K \lll 45)_L$
	k_8	$(K \lll 45)_R$
	k_9	$(K_A \lll 45)_L$
	k_{10}	$(K \lll 60)_R$
	k_{11}	$(K_A \lll 60)_L$
	k_{12}	$(K_A \lll 60)_R$
FL FL^{-1}	kl_3	$(K \lll 77)_L$
	kl_4	$(K \lll 77)_R$
Round key	k_{13}	$(K \lll 94)_L$
	k_{14}	$(K \lll 94)_R$
	k_{15}	$(K_A \lll 94)_L$
	k_{16}	$(K_A \lll 94)_R$
	k_{17}	$(K \lll 111)_L$
	k_{18}	$(K \lll 111)_R$
Whitening key	kw_3	$(K_A \lll 111)_L$
	kw_4	$(K_A \lll 111)_R$

References

[1] Martin R. Albrecht and Gregor Leander. An All-In-One Approach to Differential Cryptanalysis for Small Block Ciphers. In *Selected Areas in Cryptography (SAC'12)*, LNCS, pages 1–15. Springer, 2012.

[2] Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second Preimage Attacks on Dithered Hash Functions. In *Proceedings on Theory and Application of Cryptographic Techniques - EUROCRYPT '08*, LNCS, pages 270–288. Springer, 2008.

[3] Kazumaro Aoki, Tetsuya Ichikawa, Masayuki Kanda, Mitsuru Matsui, Shiho Moriai, Junko Nakajima, and Toshio Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In *Selected Areas in Cryptography (SAC'00)*, LNCS, pages 39–56. Springer, 2001.

[4] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and Raphael C.-W. Phan. SHA-3 Proposal BLAKE. Submission to NIST (Round 1/2), 2008. Available at `http://ehash.iaik.tugraz.at/uploads/0/06/Blake.pdf`(Accessed on 8/10/2013).

[5] Dongxia Bai and Leibo Li. New Impossible Differential Attacks on Camellia. In *Proceedings of the 8th international conference on Information Security Practice and Experience (ISPEC'12)*, LNCS, pages 80–96. Springer, 2012.

[6] Thomas Baignéres, Pascal Junod, and Serge Vaudenay. How Far Can We Go Beyond Linear Cryptanalysis? In *Proceedings on Advances in Cryptology - ASIACRYPT'04*, volume 3329 of *LNCS*, pages 432–450. Springer, 2004.

[7] Paulo S. L. M. Barreto and Vincent Rijmen. The Whirlpool Hashing Function. Submitted to NESSIE, September 2000. Revised May 2003. Available at `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html` (Accessed on 19/09/2013).

[8] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge Functions, 2007. Available at `http://www.csrc.nist.gov/pki/HashWorkshop/Public_Comments/2007_May.html`(Accessed on 08/05/2014).

[9] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak specifications. Submission to NIST (Round 2), 2009. Available at `http://keccak.noekeon.org/Keccak-specifications-2.pdf`(Accessed on 8/10/2013).

[10] Eli Biham. New Types of Cryptanalytic Attacks Using Related Keys. In *Proceedings on Theory and Application of Cryptographic Techniques EUROCRYPT '93*, LNCS, pages 398–409. Springer, 1993.

[11] Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *Proceedings on Advances in Cryptology - CRYPTO '90*, LNCS, pages 2–21. Springer, 1991.

[12] Eli Biham, Alix Biryukov, and Adi Shamir. Impossible Differential Attacks. Presented at Rump Session of CRYPTO '98, 1998.

[13] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *Proceedings on Theory and Application of Cryptographic Techniques - EUROCRYPT '99*, LNCS, pages 12–23. Springer, 1999.

[14] Alex Biryukov and Ivica Nikolić. Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In *Proceedings on Theory and Application of Cryptographic Techniques - EUROCRYPT '10*, LNCS, pages 322–344. Springer, 2010.

[15] Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolić. Distinguisher and Related-Key Attack on the Full AES-256. In *Proceedings on Advances in Cryptology - CRYPTO'09*, LNCS, pages 231–249. Springer, 2009.

[16] Céline Blondeau and Benoît Gérard. Multiple Differential Cryptanalysis: Theory and Practice. In *Proceedings of the 18th international conference on Fast software encryption (FSE'11)*, LNCS, pages 35–54. Springer, 2011.

[17] Céline Blondeau and Kaisa Nyberg. New Links between Differential and Linear Cryptanalysis. In Thomas Johansson and Phong Q. Nguyen, editors, *Proceedings on Theory and Application of Cryptographic Techniques - EUROCRYPT '13*, volume 7881 of *LNCS*, pages 388–404. Springer, 2013.

[18] Céline Blondeau, Benoît Gérard, and Kaisa Nyberg. Multiple differential cryptanalysis using LLR and χ^2. In *Proceedings of the 8th international conference on Security and Cryptography for Networks (SCN'12)*, LNCS, pages 343–360. Springer, 2012.

[19] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems (CHES 2007)*, LNCS, pages 450–466. Springer, 2007.

[20] Bruno Buchberger and Franz Winkler. *Gröbner Bases and Applications*. Cambridge University Press, 1998. ISBN 9780521632980.

[21] Don Coppersmith. The Data Encryption Standard (DES) and its strength against attacks. *IBM Journal of Research and Development*, 38(3):243–250, May 1994. ISSN 0018-8646.

[22] Nicolas Courtois, Er Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Proceedings on Theory and Application of Cryptographic Techniques - EUROCRYPT'00*, LNCS, pages 392–407. Springer, 2000.

[23] Nicolas T. Courtois and Josef Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In *Proceedings on Advances in Cryptology - ASIACRYPT'02*, LNCS, pages 267–287. Springer, 2002.

[24] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, NY, USA, 1991. ISBN 0-471-06259-6.

[25] *e-Government Recommended Ciphers List*. CRYPTREC-Cryptography Research and Evaluation Committees, 2003. `http://www.cryptrec.go.jp/english/list.html`(Accessed on 26/09/2013).

[26] Joan Daemen and Vincent Rijmen. *The Design of Rijndael -AES- The Advanced Encryption Standard*. Springer, Secaucus, NJ, USA, 2002. ISBN 3540425802.

[27] Joan Daemen and Vincent Rijmen. Probability Distributions of Correlation and Differentials in Block Ciphers. Cryptology ePrint Archive, Report 2005/212, 2005.

[28] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square. In *Proceedings of the 4th International Workshop on Fast Software Encryption (FSE '97)*, LNCS, pages 149–165. Springer, 1997.

[29] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The Block Cipher Square . In *Proceedings of the 4th International Workshop on Fast Software Encryption (FSE '97)*, LNCS, pages 149–165. Springer, 1997.

[30] Ivan Bjerre Damgård. A Design Principle for Hash Fnctions. In *Proceedings on Advances in Cryptology - CRYPTO '89*, LNCS, pages 416–427. Springer, 1989.

[31] Patrick Derbez, Pierre-Alain Fouque, and Jrmy Jean. Faster Chosen-Key Distinguishers on Reduced-Round AES. In *Progress in Cryptology - INDOCRYPT '12*, LNCS, pages 225–243. Springer, 2012.

[32] Horst Feistel. Cryptography and Computer Privacy. *Scientific American*, 228(5):15–23, May 1973.

[33] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein Hash Function Family. Submission to NIST (Round 2), 2009. Available at `http://www.skein-hash.info/sites/default/files/skein1.2.pdf`(Accessed on 8/10/2013).

[34] Federal Information Processing Standard (FIPS). Data Encryption Standard. National Bureau of Standards, U.S. Department of Commerce, 1977.

[35] Ewan Fleischmann, Christian Forler, and Stefan Lucks. McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. In *Proceedings of the 19th International Workshop on Fast Software Encryption (FSE '12)*, LNCS, pages 196–215. Springer, 2012.

[36] Pierre-Alain Fouque, Jacques Stern, and Sébastien Zimmer. Cryptanalysis of Tweaked Versions of SMASH and Reparation. In *Selected Areas in Cryptography (SAC'08)*, LNCS, pages 136–150. Springer, 2008.

[37] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl - A SHA-3 Candidate. Second Round of NIST's SHA-3 Competition, 2009. Available at `http://www.groestl.info/` (Accessed on 31/01/2013).

[38] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – A SHA-3 Candidate. A Finalist of NIST's SHA-3 Cryptographic Hash Function Competition, 2011. Available at `http://www.groestl.info/` (Accessed on 31/01/2013).

[39] Henri Gilbert and Thomas Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. In *Proceedings of the 17th international conference on Fast software encryption (FSE'10)*, LNCS, pages 365–383. Springer, 2010.

[40] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA, 2000. ISBN 0521791723.

[41] Jian Guo, San Ling, Christian Rechberger, and Huaxiong Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In *Proceedings on Advances in Cryptology - ASIACRYPT'10*, LNCS, pages 56–75. Springer, 2010.

[42] Miia Hermelin, Joo Yeon Cho, and Kaisa Nyberg. Multidimensional Extension of Matsui's Algorithm 2. In *Fast Software Encryption (FSE'09)*, LNCS, pages 209–227. Springer, 2009.

[43] Howard M. Heys. (a tutorial on linear and differential cryptanalysis.

[44] Deukjo Hong, Jaechul Sung, Seokhie Hong, Jongin Lim, Sangjin Lee, Bon-Seok Koo, Changhoon Lee, Donghoon Chang, Jesang Lee, Kitae Jeong, Hyun Kim, Jongsung Kim, and Seongtaek Chee. HIGHT: A New Block Cipher Suitable for Low-Resource Device. In *Cryptographic Hardware and Embedded Systems (CHES 2006)*, LNCS, pages 46–59. Springer, 2006.

[45] Kota Ideguchi, Elmar Tischhauser, and Bart Preneel. Improved Collision Attacks on the Reduced-Round Grøstl Hash Function. In *Proceedings of the 13th International Conference on Information Security (ISC'10)*, LNCS, pages 1–16. Springer, 2010.

[46] *ISO/IEC 18033-3, Information Technology – Security Techniques – Encryption Algorithms – Part 3: Block Ciphers*. International Organization for Standardization, 2005.

[47] Tetsu Iwata and Tadayoshi Kohno. New Security Proofs for the 3GPP Confidentiality and Integrity Algorithms. In *Proceedings of the 11th International Workshop on Fast Software Encryption (FSE '04)*, LNCS, pages 427–445. Springer, 2004.

[48] Antoine Joux and Thomas Peyrin. Hash Functions and the (Amplified) Boomerang Attack. In *Proceedings on Advances in Cryptology - CRYPTO'07*, LNCS, pages 244–263. Springer, 2007.

[49] John Kelsey, Bruce Schneier, and David Wagner. Key-Schedule Cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In *Proceedings on Advances in Cryptology CRYPTO '96*, LNCS, pages 237–251. Springer, 1996.

[50] Lars Knudsen, Gregor Leander, Axel Poschmann, and Matthew Robshaw. PRINTcipher: A Block Cipher for IC-Printing. In *Cryptographic Hardware and Embedded Systems (CHES '10)*, LNCS, pages 16–32. Springer, 2010.

[51] Lars R. Knudsen. Truncated and Higher Order Differentials. In *Fast Software Encryption (FSE'95)*, LNCS, pages 196–211. Springer, 1995.

[52] Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In *Proceedings on Advances in Cryptology - ASIACRYPT'07*, volume 4833 of *LNCS*, pages 315–324. Springer, 2007.

[53] Lars R. Knudsen and Matthew Robshaw. *The Block Cipher Companion.* Information Security and Cryptography. Springer, 2011. ISBN 978-3-642-17341-7.

[54] Xuejia Lai, James L. Massey, and Sean Murphy. Markov Ciphers and Differential Cryptanalysis. In *Proceedings on Theory and Application of Cryptographic Techniques - EUROCRYPT'91*, LNCS, pages 17–38. Springer, 1991.

[55] Mario Lamberger and Florian Mendel. Higher-Order Differential Attack on Reduced SHA-256. Cryptology ePrint Archive, Report 2011/037, 2011. `http://eprint.iacr.org/`(Accessed on 26/09/2013).

[56] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In *Proceedings on Advances in Cryptology - ASIACRYPT'09*, volume 5912 of *LNCS*, pages 126–143. Springer, 2009.

[57] Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In *Proceedings on Advances in Cryptology - ASIACRYPT'09*, LNCS, pages 126–143. Springer, 2009.

[58] Susan K. Langford and Martin E. Hellman. Differential-Linear Cryptanalysis. In *Proceedings on Advances in Cryptology - CRYPTO '94*, LNCS, pages 17–25. Springer, 1994.

[59] Gaëtan Leurent and Phong Q. Nguyen. How Risky Is the Random-Oracle Model? In *Proceedings on Advances in Cryptology - CRYPTO'09*, LNCS, pages 445–464. Springer, 2009.

[60] Leibo Li, Jiazhe Chen, and Keting Jia. New Impossible Differential Cryptanalysis of Reduced-Round Camellia. In *Cryptology and Network Security (CANS '11)*, LNCS, pages 26–39. Springer, 2011.

[61] Chae Hoon Lim. A Revised Version of Crypton - Crypton V1.0. In *Proceedings of the 6th International Workshop on Fast Software Encryption (FSE '99)*, LNCS, pages 31–45. Springer, 1999.

[62] Shusheng Liu, Zheng Gong, and Libin Wang. Improved Related-Key Differential Attacks on Reduced-Round LBlock. In *Proceedings of the 14th international conference on Information and Communications Security(ICICS'12)*, LNCS, pages 58–69. Springer, 2012.

[63] Ya Liu, Dawu Gu, Zhiqiang Liu, and Wei Li. Impossible Differential Attacks on Reduced-Round LBlock. In *Proceedings of the 8th international conference on Information Security Practice and Experience (ISPEC'12)*, LNCS, pages 97–108. Springer, 2012.

[64] Ya Liu, Leibo Li, Dawu Gu, Xiaoyun Wang, Zhiqiang Liu, Jiazhe Chen, and Wei Li. New Observations on Impossible Differential Cryptanalysis of Reduced-Round Camellia. In *Proceedings of the 19th international conference on Fast Software Encryption (FSE'12)*, LNCS, pages 90–109. Springer, 2012.

[65] Jiqiang Lu, Yongzhuang Wei, Pierre-Alain Fouque, and Jongsung Kim. Cryptanalysis of Reduced Versions of the Camellia Block Cipher. *Information Security, IET*, 6(3):228–238, 2012. ISSN 1751-8709. doi: 10.1049/iet-ifs.2011.0342.

[66] Jiqiang Lu, Yongzhuang Wei, Enes Pasalic, and Pierre-Alain Fouque. Meet-in-the-Middle Attack on Reduced Versions of the Camellia Block Cipher. In *Advances in Information and Computer Security (WSEC '12)*, LNCS, pages 197–215. Springer, 2012.

[67] Hamid Mala, Mohsen Shakiba, Mohammad Dakhilalian, and Ghadamali Bagherikaram. New Results on Impossible Differential Cryptanalysis of ReducedRound Camellia128. In *Selected Areas in Cryptography (SAC '09)*, LNCS, pages 281–294. Springer, 2009.

[68] James Massey, Gurgen Khachatrian, and Melsik Kuregian. Nomination of SAFER++ as Candidate Algorithm for the New European Schemes for Signatures, Integrity, and Encryption (NESSIE). First Open NESSIE Workshop, 2000.

[69] Mitsuru Matsui. The First Experimental Cryptanalysis of the Data Encryption Standard. In *Proceedings on Advances in Cryptology - CRYPTO '94*, LNCS, pages 1–11. Springer, 1994.

[70] Mitsuru Matsui. Linear Cryptanalysis of the Data Encryption Standard. In *Proceedings on Theory and Application of Cryptographic Techniques - EUROCRYPT'93*, LNCS, pages 386–397. Springer, 1994.

[71] Mitsuru Matsui and Atsuhiro Yamagishi. A New Method for Known Plaintext Attack of FEAL Cipher. In *Proceedings on Theory and Application of Cryptographic Techniques - EUROCRYPT'92*, LNCS, pages 81–91. Springer, 1992.

[72] Florian Mendel, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In *Selected Areas in Cryptography (SAC'09)*, LNCS, pages 16–35. Springer, 2009.

[73] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In *Fast Software Encryption (FSE'09)*, volume 5665 of *LNCS*, pages 260–276. Springer, 2009.

[74] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Rebound Attacks on the Reduced Grøstl Hash Function. In *The Cryptographers' Track at the RSA Conference(CT-RSA)*, LNCS, pages 350–365. Springer, 2010.

[75] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996. ISBN 0849385237.

[76] Ralph C. Merkle. Protocols for Public Key Cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–134. IEEE Computer Society Press, 1980.

[77] Ralph C. Merkle. One way hash functions and DES. In *Proceedings on Advances in Cryptology - CRYPTO '89*, LNCS, pages 428–446. Springer, 1989.

[78] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, 1979.

[79] Marine Minier and María Naya-Plasencia. A Related Key Impossible Differential Attack Against 22 Rounds of the Lightweight Block Cipher LBlock. *Information Processing Letters*, 112(16):624–629, 2012.

[80] Marine Minier, Raphael C. Phan, and Benjamin Pousse. Distinguishers for Ciphers and Known Key Attack against Rijndael with Large Blocks. In *Proceedings on Cryptology in Africa - AFRICACRYPT'09*, LNCS, pages 60–76. Springer, 2009.

[81] National Institute of Standards and Technology (NIST). *FIPS 180-1:Secure Hash Standard*, 1995. Available at `http://www.itl.nist.gov/fipspubs/fip180-1.htm` (Accessed on 15/07/2013).

[82] National Institute of Standards and Technology (NIST). *FIPS 180-2:Secure Hash Standard*, 2002. Available at `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf` (Accessed on 15/07/2013).

[83] *Final Report of European Project IST-1999-12324*. NESSIE-New European Schemes for Signatures, Integrity, and Encryption, 1999. `https://www.cosic.esat.kuleuven.be/nessie/Bookv015.pdf`(Accessed on 26/09/2013).

[84] Ivica Nikolić, Josef Pieprzyk, Przemyslaw Sokolowski, and Ron Steinfeld. Known and Chosen Key Differential Distinguishers for Block Ciphers. In *Proceedings of the 13th International Conference on Information Security and Cryptology (ICISC'10)*, LNCS, pages 29–48. Springer, 2011.

[85] Thomas Peyrin. Improved Differential Attacks for ECHO and Grøstl. In *Proceedings on Advances in Cryptology - CRYPTO'10*, LNCS, pages 370–392. Springer, 2010.

[86] Josef Pieprzyk and Babak Sadeghiyan. *Design of Hashing Algorithms*. Springer New York, Inc., Secaucus, NJ, USA, 2001. ISBN 0387575006.

[87] Thomas Pornin. Email listing of www.crypto.stackexchange.com, 2011. http://crypto.stackexchange.com/questions/1173/what-is-pre-image-resistance-and-how-can-the-lack-thereof-be-exploited (Accessed on 31/01/2013).

[88] Bart Preneel. Cryptographic Hash Functions: Theory and Practice. Invited Talk at INDOCRYPT 2010, 2010.

[89] Ronald L. Rivest. The MD4 Message Digest Algorithm. In *Proceedings on Advances in Cryptology - CRYPTO '90*, pages 303–311. Springer, 1991.

[90] Ronald L. Rivest. The MD5 Message Digest Algorithm. Request For Comments (RFC) 1321, 1992.

[91] Yu Sasaki. Known-Key Attacks on Rijndael with Large Blocks and Strengthening Shiftrow Parameter. In *Proceedings of the 5th international conference on Advances in information and computer security (IWSEC'10)*, LNCS, pages 301–315. Springer, 2010.

[92] Yu Sasaki and Lei Wang. Comprehensive Study of Integral Analysis on 22-Round LBlock. In *Proceedings of the 15th International Conference Information Security and Cryptology (ICISC 2012)*, LNCS, pages 156–169. Springer, 2013.

[93] Yu Sasaki and Kan Yasuda. Known-Key Distinguishers on 11-Round Feistel and Collision Attacks on Its Hashing Modes. In *Fast Software Encryption (FSE'11)*, volume 6733 of *LNCS*, pages 397–415. Springer, 2011.

[94] Yu Sasaki, Yang Li, Lei Wang, Kazuo Sakiyama, and Kazuo Ohta. Non-full-active Super-Sbox Analysis: Applications to ECHO and Grøstl. In *Proceedings on Advances in Cryptology - ASIACRYPT'10*, LNCS, pages 38–55. Springer, 2010.

[95] Ali Aydm Selçuk and Ali Biçak. On Probability of Success in Linear and Differential Cryptanalysis. In *Proceedings of the 3rd international conference on Security in communication networks - SCN'02*, LNCS, pages 174–185. Springer, 2003.

[96] Claude Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4):656–715, October 1949.

[97] Akihiro Shimizu and Shoji Miyaguchi. Fast Data Encipherment Algorithm FEAL. In *Proceedings on Theory and Application of Cryptographic Techniques EUROCRYPT '87*, LNCS, pages 267–278. Springer, 1987.

[98] Richard Singleton. Maximum Distance q-nary Codes. *IEEE Transactions on Information Theory*, 10(2):116–118, September 1964. ISSN 0018-9448.

[99] Hadi Soleimany and Kaisa Nyberg. Zero-Correlation Linear Cryptanalysis of Reduced-Round LBlock. In *International Workshop on Coding and Cryptography (WCC 2013)*, 2013.

[100] R. Brandner T.Gondrom and U. Pordesch. Evidence Record Syntax. RFC 4998, 2007.

[101] Paul C. van Oorschot and Michael J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *Journal of Cryptology*, 12:1–28, 1999.

[102] 3GPP TS 35.201 version 3.1.1. Specification of the 3GPP Confidentiality and Integrity Algorithms, Document 1: $f8$ and $f9$ Specification, 1999. Available at `http://www.3gpp.org/DynaReport/35201.htm`(Accessed on 08/05/2014).

[103] 3GPP TS 35.202 version 3.1.2. Specification of the 3GPP Confidentiality and Integrity Algorithms, Document 2: KASUMI Algorithm Specification, 1999. Available at `http://www.3gpp.org/DynaReport/35202.htm`(Accessed on 08/05/2014).

[104] David Wagner. The Boomerang Attack. In *Proceedings of the 6th International Workshop on Fast Software Encryption (FSE '99)*, LNCS, pages 156–170. Springer, 1999.

[105] David Wagner. A Generalized Birthday Problem. In *Proceedings on Advances in Cryptology - CRYPTO'02*, LNCS, pages 288–303. Springer, 2002.

[106] Ralf-Philipp Weinmann. *Algebraic Methods in Block Cipher Cryptanalysis*. PhD thesis, 2009.

[107] David Wheeler and Roger Needham. TEA, a Tiny Encryption Algorithm. In *Fast Software Encryption (FSE'94)*, LNCS, pages 363–366. Springer, 1994.

[108] Hongjun Wu. The Hash Function JH. Submission to NIST (Round 1/2), 2009. Available at `http://ehash.iaik.tugraz.at/uploads/1/1d/Jh20090915.pdf`(Accessed on 8/10/2013).

[109] Shuang Wu, Dengguo Feng, Wenling Wu, Jian Guo, Le Dong, and Jian Zou. (Pseudo) Preimage Attack on Round-Reduced Grøstl Hash Function and Others. In *Fast Software Encryption (FSE'12)*, LNCS, pages 127–145. Springer, 2012.

[110] Wenling Wu and Lei Zhang. LBlock: A Lightweight Block Cipher. In *Applied Cryptography and Network Security (ACNS '11)*, LNCS, pages 327–344. Springer, 2011.

[111] Jian Zou, Wenling Wu, Shuang Wu, and Le Dong. Improved (Pseudo) Preimage Attack and Second Preimage Attack on Round-Reduced Grøstl. Cryptology ePrint Archive, Report 2012/686, 2012. `http://eprint.iacr.org/`(Accessed on 22/05/2014).