Analysis and Design of Symmetric Cryptographic Algorithms

Isobe, Takanori

Doctoral Dissertation

Analysis and Design of Symmetric Cryptographic Algorithms

共通鍵暗号アルゴリズムの解析と設計

July 2013

Graduate School of Engineering,
Kobe University

Takanori Isobe

Doctoral Dissertation

Analysis and Design of Symmetric Cryptographic Algorithms

共通鍵暗号アルゴリズムの解析と設計

July 2013

Graduate School of Engineering,
Kobe University

Takanori Isobe

Acknowledgement

I would like to thank everyone who contributed to this thesis. First of all, I would like to express my sincere gratitude to my superior Prof. Masakatsu Morii for introducing me to *cryptanalysis of symmetric cipher* as my research topic. He gave me respect to do the research in my own way. It helps me to establish the basis of my career in this field. I also would like to thank the judging committee members, Prof. Sumio Masuda and Prof. Yasuyoshi Yasaka for reviewing this manuscript and giving me valuable feedbacks. I would like to thank Prof. Hidenori Kuwakado and Assistant Prof. Minoru Kuribayashi for giving me helpful advice and supporting my research in Kobe University.

I would like to thank Kyoji Shibutani. When I joined Sony Corporation in 2008, I did not have enough ability as a cryptographic researcher. He taught me a lot of things, not only about symmetric cryptographic ciphers but also basic things like how to write academic articles, find an interesting problem, and approach it. He guided me in a good direction through collaborative works. I am grateful to co-authors of Sony: Toru Akishita, Harunaga Hiwatari, Ji Li, Atsushi Mitsuda and Taizo Shirai for insightful discussions. I have learned much from them through the team-work. I would like to thank Asami Mizuno and Toru Akishita for giving me a chance to get this PhD. Many results included in this manuscript have been done in Sony. I am also thankful to all the members in same division for constant support.

I am grateful to all the members of a stream cipher research group of Kobe University: Toshihiro Ohigashi, Yuhei Watanabe, Atsushi Nagao and Tsubasa Tsukaune. We spent plenty of wonderful time on discussing the attacks on stream ciphers, especially RC4.

I am also thankful co-authors Jiageng Chen and Yu Sasaki for interesting discussions at ASK 2012 and writing papers. It is very exciting for me to collaborate external researches.

I greatly appreciate my parents, grandparents and my sister for their longtime support. Finally, I express deep gratitude to my wife Mio for her constant encouragement and unconditional help without which the thesis would never have been possible.

Takanori Isobe
Tokyo, July 2013

Abstract

Cryptography is the fundamental technology for the information security. Based on mathematical aspects of cryptography such as a randomness, a one-wayness, and an unpredictability, a number of information systems, electrical appliances and network services ensure its security. Among them, *symmetric cryptographic algorithms* including block ciphers, stream ciphers, hash functions and message authenticated codes play a central roll in today's information society. These algorithms mainly provide a data confidentiality and a data integrity, and are efficiently implementable in the software and the hardware. Actually, 99 % of encrypted date around the world is encrypted with symmetric cryptographic algorithms.

This thesis is dedicated to *analysis and design of symmetric cryptographic algorithms.* We deal with security analysis of block ciphers, stream ciphers and hash functions, and provide new insights and feedbacks for designing secure algorithms. We introduce a block cipher *Piccolo* as a secure block cipher.

Analysis of Block Cipher

Block cipher is a permutation that converts a fixed-length input (a plaintext) into a fixed-length output (a ciphertext), depending on a secret key. Since the confidentiality of a key guarantees the security of block ciphers, the computational difficulty of finding a key is the important criteria for its security evaluations. This thesis mainly tackles enhancements of the Meet-in-the-Middle (MITM) attack, which is a powerful technique for key recovery attacks of block ciphers.

To begin with, we develop a new variant of the MITM attack called Reflection-Meet-in-the-Middle (R-MITM) attack, and apply it to the full GOST block cipher, which is the Russian encryption standard block cipher. We propose a $first$ single-key attack on the full-round GOST block cipher.

We apply the MITM attack to lightweight block ciphers Piccolo, LED and XTEA by exploiting its slow diffusion property and low key dependency. As a results, we update best attacks of these ciphers with respect to the number of attack rounds in the single key setting.

We extend the MITM attack to apply it to wider class block ciphers. We propose an all subkey recovery approach which aims to recover all subkeys instead of the master key. It allows us to construct MITM attacks without dealing with the key scheduling function. Then we can update the best single-key attacks on SHACAL-2, CAST, KATAN, Blowfish and FOX with respect to the number of attack rounds.

Also, we construct related-key boomerang attacks, which use statistical weakness, on the KATAN block cipher, and improve best attacks.

Analysis of Stream Cipher

Stream cipher is a function that converts fixed-length inputs (a key and an Initial Vector (IV)) into an arbitrary-length output called keystream. A ciphertext is obtained by XORing a plaintext and a keystream. This thesis analyzes the security of stream ciphers Py, RAKAPOSHI and RC4.

Py is a software oriented stream cipher, and was submitted to the ECRYPT stream cipher project (eSTREAM). We show a practical key recovery attack of Py, utilizing a guess and determine approach in conjunction with an IV collision property.

RAKAPOSHI is a hardware oriented lightweight stream cipher. We reveal that RAKAPOSHI has a slide property such that two different Key-IV pairs generate the same keystream but n-bit shifted. Then, we show that this property is exploitable for a key recovery attacks on RAKAPOSHI in the related-key setting.

RC4 is a software oriented stream cipher, and is one of most widely used stream ciphers in the world. We introduce new biases in initial bytes of the keystrem of RC4, and demonstrate a plaintext recovery attack using these biases in the broadcast setting where the same plaintext is encrypted with different user keys. Also, we show that our set of these biases are applicable to plaintext recovery attacks, key recovery attacks and distinguishing attacks.

Analysis of Hash Function

Hash function is a function that converts an arbitrary-length input (a message) into a fixed-length output (a digest) without using a secret key. A cryptographic hash function is required to satisfy at least three security properties: preimage resistance, second preimage resistance and collision resistance. This thesis focuses on the MITM technique of hash functions, and evaluate the security of well known hash functions SHA-2, Tiger and Skein.

We propose one-block preimage attacks on reduced Tiger and SHA-2 by using new technique, called independent transform, for finding neutral words for the MITM technique. We improve previous preimage attack of Tiger and SHA-2.

We provide a new insight of relation between a meet-in-the-middle preimage attack and a pseudo collision attack, and presents a generic technique to convert a MITM preimage attacks into pseudo collision attacks. By using our technique, we update best attacks of pseudo collision attacks of SHA-256, SHA-512 and Skein with respect to the number of attack rounds.

Design of Block Cipher

Finally we gives several feedbacks for secure designs of symmetric cryptographic algorithms. Adopting secure design criteria, we introduce a 64-bit blockcipher *Piccolo* supporting 80 and 128-bit keys as a secure block cipher. We show that *Piccolo* offers a sufficient security level against known analyses including recent related-key differential attacks and meet-in-the-middle attacks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Cryptography is the science of the secret, and the core technology for the information security. With development of the highly-networked information society, cryptography becomes increasingly important as the way of protecting sensitive information, controlling the access to the system and authenticating the data. As a results, we can see cryptography everywhere, from rich-resourced cloud servers to resource-constrained devices such as RFIDs and sensor nodes. Based on mathematical aspects of cryptography such as a randomness, a one-wayness and an unpredictability, today's secure system, products, network services are constructed, e.g., an internet banking, an internet shopping, an electronic voting and a secure multi-party computation.

Modern cryptography is based on a *computational security*, not an informational security which is due to the fact that an adversary does not have enough information to break it, regardless of the computation power. In contrast, the computational security relies on the computational in-feasibility and difficulty for violating security requirements. If an adversary has enough computational power, modern cryptographic schemes can be broken. However the approach of the computational security allows us to construct today's practical cryptographic schemes with reliable security evaluations, while the informational security is of less practical, e.g., it requires large amount of secret information to achieve its security.

1.1.1 Symmetric Cryptographic Algorithms

Symmetric cryptographic algorithms play a central roll in the security of today's computer-aimed networks and systems. In the symmetric setting (often called private key setting or secret key setting), two parties in advance share some secret information (*key*), and use it to communicate secretly with each other. Since the same key is used in both sides, it is called symmetric cryptographic algorithm. In general, these algorithms include block ciphers, stream ciphers, hash functions and message authenticated codes.

A block cipher and a stream cipher provide a data *confidentiality*. In these functions, a *plaintext* is converted into a scrambled data *ciphertext*, depending on a secret key, and the plaintext is recovered from the ciphertext with same secret key. These processes are referred to as *encryption* and *decryption*, respectively. For example, a block cipher and a stream cipher are used in the case where two parties try to share a data secretly through public communication channels such as an Internet. The sender encrypts the data to be sent with the shared key. After the sender sends it to the receiver, the receiver decrypts it with the same key to obtain the plaintext. Then the sender and the receiver successfully share the data. Formally, a block cipher is a permutation that converts a fixed-length input (a plaintext) into a fixed-length output (a ciphertext), depending on a secret key. A stream cipher is a function that converts fixed-length inputs (a key and an Initial Vector (IV)) into an arbitrary-length output called keystream. A ciphertext is obtained by XORing a plaintext and a *keystream*.

Message authentication code (called MAC) algorithm provide a data *integrity*. The sender computes an authentication tag from the data with a secret key, and send it to the receiver with data. Then, the receiver computes the authentication tag corresponding to the message in the same manner. If the authentication tag calculated by the receiver coincides with the one obtained from the communication channel, the message is considered as valid, and the integrity of data is confirmed. Formally, a MAC is a function that converts an arbitrary-length input (message) into a fixed-length output (digest) by using a secret key.

Hash functions are similar to message authentication codes and can also be used to provide a data integrity. Although a hash function does not use a secret key, it is often regarded as a family of symmetric cryptographic algorithms, because the design principle is very similar to it. A cryptographic hash function is required to satisfy at least three security properties: preimage resistance, second preimage resistance and collision resistance. Formally, a hash function is a function that converts an arbitrary-length input (message) into a fixed-length output (digest).

1.1.2 Cryptanalysis

Design of modern cryptographic algorithms follows the Kerckhoffs's principle :

The cipher method must not be required to be secret,
and it must be able to fall into the hands of the enemy without inconvenience.

This principle says that cryptographic algorithms should not be kept secret, and only a key should be kept secret. In other words, cryptographic algorithms should be designed so as to be secure even if an adversary knows the details of all the component algorithm of the scheme, as long as the adversary does not know the key.

Cryptanalysis of Symmetric Cryptographic Algorithms

According to the Kerckhoffs's principle, the details of algorithms are given, and only a key is unknown. *Cryptanalysis* is to analyze cryptographic algorithms for finding mathematical weaknesses of it. To put it more precisely, the purpose of cryptanalysis is to study *attacks* for violating security claims of cryptographic algorithms, e.g., a randomness of outputs, a computational difficulty of finding keys and forging a MAC tag, a collision resistance and a preimage resistance. Following the policy of the open cryptographic design, in general, after an cryptographic algorithm is public, and then third party researchers analyze the algorithm and try to break it. If no weakness is found for a long time, it is regarded as a secure algorithm. Actually, AES [1] and SHA-3 [2], which are the American standard block cipher and the hash function, were selected through such an open evaluation process. Over 12 years after the end of AES project in 2001, no weakness is found so far (July, 2013). Thus, the cryptographic community believes that the process of the open cryptographic design and the evaluation are well suited for selecting secure algorithms, from the academical and the industrial points of views. Therefore, cryptanalysis is one of the important field in cryptography, along with design of cryptographic algorithms.

Techniques of Cryptanalysis

After pioneer works of a differential cryptanalysis by Biham and Shamir [3] in 1990 and a linear cryptanalysis [4] by Matsui in 1993 for block ciphers, a number cryptanalysis methods and technique were developed in two decades, e.g., truncated differential attack [5], higher order differential attack [5], related key attack [6], impossible differential attack [7], boomerang attack [8], meet-in-the middle attack [9], slide attack [10] and integral attack [11]. Recently, attack techniques for hash functions have made tremendous progress after Wang's breakthrough results [12, 13]. Rebound attack [14], rotational attack [15] and meet-in-the-middle preimage attack [16] are developed during the SHA-3 competition.

1.2 Our Contribution and Outline

This thesis is dedicated to *analysis and design of symmetric cryptographic algorithms.* We deal with security analysis of block ciphers, stream ciphers and hash functions, and provide new insights and feedbacks for designing secure algorithms.

First, we introduce definitions and security requirements of symmetric cryptographic algorithms. The main part of this thesis consists of four parts : analysis of block cipher, analysis of stream cipher, analysis of hash function and design of block cipher. The details of the main part are as follows.

Analysis of Block Cipher

- **Chapter 3** shows a first single-key attack on full GOST block cipher. The GOST block cipher is the Russian encryption standard published in 1989 [17]. In spite of considerable cryptanalytic efforts over the past 20 years, a key recovery attack on the full GOST block cipher without any key conditions (*e.g.*, weak keys and related keys) has not been published yet. In this chapter, we show the first single-key attack, which works for all key classes, on the full GOST block cipher. At first, we develop a new attack framework called *Reflection-Meet-in-the-Middle Attack.* This approach combines techniques of the reflection attack [18] and the meet-in-the-middle (MITM) attack [9]. Then, we apply it to the GOST block cipher employing bijective S-boxes. In order to construct the full-round attack, we use additional novel techniques which are the effective MITM techniques using equivalent keys on a small number of rounds. As a result, a key can be recovered with a time complexity of 2^{225} encryptions and 2^{32} known plaintexts. Moreover, we show that our attack is applicable to the full GOST block cipher using any S-boxes, including non-bijective S-boxes.

- **Chapter 4** investigate the security of the lightweight block ciphers against the meet-in-the-middle (MITM) attack [9]. Since the MITM attack mainly exploits low key-dependency in a key expanding function, the block ciphers having a simple key expanding function are likely to be vulnerable to the MITM attack. On the other hand, such a simple key expanding function leads compact implementation, and thus is utilized in several lightweight block ciphers. However, the security of such lightweight block ciphers against the MITM attack has not been studied well so far. We apply the MITM attack to the XTEA [19], LED [20] and Piccolo [21], then give more accurate security analysis for them. Specifically, combining thorough analysis with new techniques, we present the MITM attacks on 29, 8, 16, 14 and 21 rounds of XTEA, LED-64, LED-128, Piccolo-80 and Piccolo-128, respectively. Consequently, it is demonstrated that the MITM attack is the most powerful attack in the single-key setting on those ciphers with respect to the number of attacked rounds.

- **Chapter 5** extends the MITM attack so that it can be applied to a wider class of block ciphers. In our approach, MITM attacks on block ciphers consisting of a complex key schedule can be constructed. We regard all subkeys as independent variables, then transform the game that finds the user-provided key to the game that finds all independent subkeys. We apply our approach called all subkeys recovery (ASR) attack to block ciphers employing a complex key schedule such as CAST-128, [22] SHACAL-2 [23], KATAN [24], Blowfish [25] and FOX [26], and present the best attacks on them with respect to the number of attacked rounds in literature. Moreover, since our attack is simple and generic, it is applied to the block ciphers consisting of any key schedule functions even if the key schedule is an ideal function.

- **Chapter 6** analyzes the KATAN block cipher. KATAN/KTANTAN is a family of hardware oriented block ciphers proposed at CHES 2009 [24] . Although the KTANTAN family have been broken by a meet-in-the-middle approach [9], the KATAN family are secure at present. In this chapter, we investigate the KATAN family in the related-key boomerang framework [27] with several techniques. By using an efficient differential characteristics search method, long boomerang distinguishers can be built. Furthermore, the key recovery phase is optimized by exploiting several properties of the round function such as the high linearity of the round function and the slow key diffusion. As a result, we can attack 174, 145 and 130 rounds of KATAN32, KATAN48 and KATAN64, which substantially improve the known best results whose attacked rounds are 120, 103, 94 rounds, respectively. Our attacks are confirmed by various experimental verifications, especially, we give concrete right quartets for KATAN32.

Analysis of Stream Cipher

- **Chapter 7** proposes an effective key recovery attack on stream ciphers Py [28] and Pypy [29] with chosen IVs. Our method uses an internal-state correlation based on the vulnerability that the randomization of the internal state in the KSA is inadequate, and it improves two previous attacks proposed by Wu and Preneel (a WP-1 attack and a WP-2 attack) [30, 31]. For a 128-bit key and a 128-bit IV, the WP-1 attack can recover a key with 2^{23} chosen IVs and time complexity 2^{72}. First, we improve the WP-1 attack by using the internal-state correlation (called a P-1 attack). For a 128-bit key and a 128-bit IV, the P-1 attack can recover a key with 2^{23} chosen IVs and time complexity 2^{48}, which is $1/2^{24}$ of that of the WP-1 attack. The WP-2 attack is another improvement on the WP-1 attack, and it has been known as the best previous attack against Py and Pypy. For a 128-bit key and a 128-bit IV, the WP-2 attack can recover a key with 2^{23} chosen IVs and time complexity 2^{24}. Second, we improve the WP-2 attack by using the internal-state correlation as well as the P-1 attack (called a P-2 attack). For a 128-bit key and a 128-bit IV, the P-2 attack can recover a key with 2^{23} chosen IVs and time complexity 2^{24}, which is the same capability as that of the WP-2 attack. However, when the IV size is from 64 bits to 120 bits, the P-2 attack is more effective than the WP-2 attack. Thus, the P-2 attack is the known best attack against Py and Pypy.

- **Chapter 8** gives a first security evaluation of a lightweight stream cipher RAKAPOSHI [32]. In particular, we analyze a slide property of RAKAPOSHI such that two different Key-IV pairs generate the same keystream but n-bit shifted. To begin with, we demonstrate that any Key-IV pair has a corresponding *slide* Key-IV pair that generates an n-bit shifted keystream with probability of 2^{-2n}. In order to experimentally support our results, some examples of such pairs are given. Then, we show that this property is able to be converted into key recovery attacks on RAKAPOSHI. In the related-key setting, our attack based on the slide property can recover a 128-bit key with time complexity of 2^{41} and 2^{38} chosen IVs. Moreover, by using a variant of slide property called partial slide pair, this attack is further improved, and then a 128-bit key can be recovered with time complexity of 2^{33} and 2^{30} chosen IVs. Finally, we present a method for speeding up the brute force attack by a factor of 2 in the single key setting.

- **Chapter 9** investigates the practical security of RC4 in broadcast setting where the same plaintext is encrypted with different user keys. We introduce several new biases in the initial (1st to 257th) bytes of the RC4 keystream, which are substantially stronger than known biases [33, 34, 35, 36]. Combining the new biases with the known ones, a cumulative list of strong biases in the first 257 bytes of the RC4 keystream is constructed. We demonstrate a plaintext recovery attack using our strong bias set of initial bytes by the means of a computer experiment. Almost all of the first 257 bytes of the plaintext can be recovered, with probability more than 0.8, using only 2^{32} ciphertexts encrypted by randomly-chosen keys. We also propose an efficient method to extract later bytes of the plaintext, after the 258th byte. The proposed method exploits our bias set of first 257 bytes in conjunction with the digraph repetition bias proposed by Mantin in EUROCRYPT 2005 [37], and sequentially recovers the later bytes of the plaintext after recovering the first 257 bytes. Once the possible candidates for the first 257 bytes are obtained by our bias set, the later bytes can be recovered from about 2^{34} ciphertexts with probability close to 1. Finally, we show that our set of these biases are applicable to key recovery attacks and distinguishing attacks.

Analysis of Hash Function

- **Chapter 10** shows new preimage attacks on reduced Tiger [38] and SHA-2 [39]. Our new preimage attack finds a one-block preimage of Tiger reduced to 16 rounds with a complexity of 2^{161} [40]. The proposed attack is based on meet-in-the-middle attacks by Aoki and Sasaki [16]. It seems difficult to find "independent words" of Tiger at first glance, since its key schedule function is much more complicated than that of MD4 or MD5. However, we developed techniques to find independent words efficiently by controlling its internal variables. Surprisingly, the similar techniques can be applied to SHA-2 including both SHA-256 and SHA-512. We present a one-block preimage attack on SHA-256 and SHA-512 reduced to 24 (out of 64 and 80) steps with a complexity of 2^{240} and 2^{480}, respectively. To the best of our knowledge, our attack is the best known preimage attack on reduced-round Tiger and our preimage attack on reduced-step SHA-512 is the first result. Furthermore, our preimage attacks can also be extended to second preimage attacks directly, because our attacks can obtain random preimages from an arbitrary IV and an arbitrary target.

- **Chapter 11** presents a new technique to construct a collision attack from a particular preimage attack which is called a partial target preimage attack. Since most of the recent meet-in-the-middle preimage attacks can be regarded as the partial target preimage attack, a collision attack is derived from the meet-in-the-middle preimage attack [16]. By using our technique, pseudo collisions of the 43-step reduced SHA-256 and the 46-step reduced SHA-512 can be obtained with complexities of 2^{126} and $2^{254.5}$, respectively. As far as we know, our results are the best pseudo collision attacks on both SHA-256 and SHA-512 in literature. Moreover, we show that our pseudo collision attacks can be extended to 52 and 57 steps of SHA-256 and SHA-512, respectively, by combined with the recent preimage attacks on SHA-2 by bicliques. Furthermore, since the proposed technique is quite simple, it can be directly applied to other hash functions. We apply our algorithm to several hash functions including Skein which is the SHA-3 finalists [41]. We present not only the best pseudo collision attacks on SHA-2 family, but also a new insight of relation between a meet-in-the-middle preimage attack and a pseudo collision attack.

Design of Block Cipher

- **Chapter 12** gives several feedbacks for secure designs of symmetric cryptographic algorithms. Then, we introduce a 64-bit blockcipher *Piccolo* supporting 80 and 128-bit keys. Adopting secure design criteria, *Piccolo* achieves high security. We show that *Piccolo* offers a sufficient security level against known analyses including recent related-key differential attacks [6, 27] and meet-in-the-middle attacks [9].

Chapter 2

Symmetric Cryptographic Algorithms

Symmetric cryptographic algorithms are divided into block ciphers, stream ciphers, hash functions and a message authenticated codes. A block cipher and a stream cipher provide a *confidentiality*, and a hash function and a message authenticated code provide an *integrity*. In this chapter, we explain definitions, security requirements, attack scenarios of block ciphers, stream ciphers and hash functions, respectively.

2.1 Block Cipher

A block cipher is a permutation that converts a fixed-length input (a plaintext) into a fixed-length output (a ciphertext), depending on a secret key, and provides a data *confidentiality* as shown in Fig. 2.1. Formally it is defined as follows.

Definition 1. *Block cipher is a keyed permutation such that a mapping $E_k : \{0,1\}^k \times \{0,1\}^n = \{0,1\}^n$, where n is block size and k is key size. The inverse of E_k is defined as E_k^{-1}. If a key is fixed, the mapping is bijective.*

A block ciphers is a n-bit to n-bit keyed permutation. There are $(2^n)!$ permutations in a set of n bit permutations. A key plays a role to chooses one permutation from $(2^n)!$ permutations. Thus, a n-bit block and k-bit key block cipher consists of 2^k n-bit permutations.

Mode of operation is a way to encrypt an arbitrary length input, based on block ciphers. We introduce major mode of operations.

Electronic Codebook (ECB) mode : This is the simplest mode of operation. Given blocks of a plaintext, each block is separately encrypted with the same key. Outputs obtained from each block are concatenated, and it is regarded as a ciphertext. This mode can be parallel implemented for each block. This mode is described in Fig. 2.2.

Cipher Block Chaining (CBC) mode: In this mode of operation, every block of a plaintext is XORed with the previous ciphertext block. The result is then used as an input to the block cipher. The first block of a plaintext is XORed with a public initial value (IV), where IV should be randomly chosen

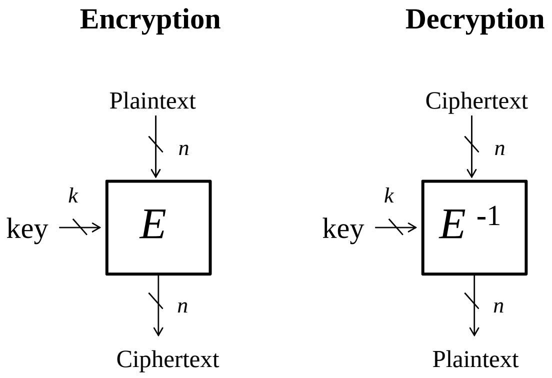


Figure 2.1: Block cipher of n-bit block and k-bit key

and be unpredictable for ensuring its security. This encryption mode needs to be serially implemented, and the parallel implementation is infeasible. Note that the decryption mode is parallel executable with the previous block of a ciphertext. This mode is described in Fig. 2.3 .

Counter (CTR) mode : In this mode of operation, a counter value is encrypted by the block cipher, and the output is XORed with the block of a plaintext to obtain the block of a ciphertext. The counter is incremented before it is used in the next block. This mode can be seen as a way of generating a pseudo random number from block ciphers. This mode is fully parallel implemented in both the encryption and decryption mode. Also it is possible to generate a pseudo random number before a plaintext or a ciphertext is given, independently from inputs. This mode is described in Fig. 2.4.

Cipher Feedback (CFB) mode : In this mode of operation, the previous block of a ciphertext is used as an input to the block cipher. A ciphertext is obtained by XORing the output of a block cipher and the block of a plaintext. The input of the first block is a public initial value (IV). The parallel implementation is infeasible in the encryption mode similar to the CBC mode. This is described in Fig. 2.5.

Output Feedback (OFB) mode: In this mode of operation, the previous output of a block cipher is used as an input to the block cipher. A ciphertext is obtained by XORing the output of a block cipher and the block of a plaintext. Similar to the CTR mode, it looks like a way of generating a pseudo random number by using block ciphers. It is also possible to generate a pseudo random number a head of time, independently from inputs, while the parallel implementation is infeasible in the encryption mode. This mode is described in Fig. 2.6.

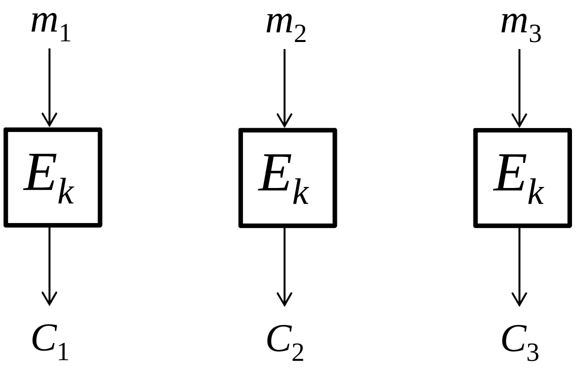


Figure 2.2: Electronic Codebook (ECB) mode

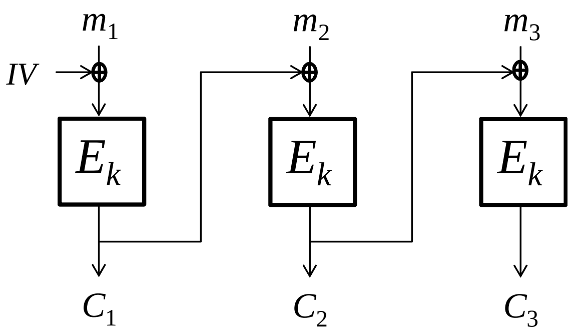


Figure 2.3: Cipher Block Chaining (CBC) mode

2.1.1 Security Requirement

A block cipher should be modeled as a *(strong) pseudo random permutation.* This enables a formal analysis of block cipher based constructions such as mode of operations, message authentication codes, authentication protocols and hash functions.

Before definition of a (strong) pseudo random permutation, we define the term of *negligible* as follows.

Definition 2. [42] *A function f is negligible if for every polynomial $p(\cdot)$ there exist an N such that for all integers $n > N$ it holds that $f(n) < 1/p(n)$.*

Such a function f is defined as `negl`. If the probability that the algorithm is broken is less than $1/p(n)$ for every polynomial p, the algorithm is regarded as *secure* in terms of the computational security.

The definition of the *pseudo random permutation* is given as follows.

Definition 3. [42] *Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, keyed permutation. We say that F is a pseudo permutation if for all probabilistic polynomial-time distinguisher D, there exist a negligible function* `negl` *such that*

$$\left| Pr[D^{F_k(\cdot)}(1^n)] - Pr[D^{f(\cdot)}(1^n)] \right| \le \mathtt{negl}(n)$$

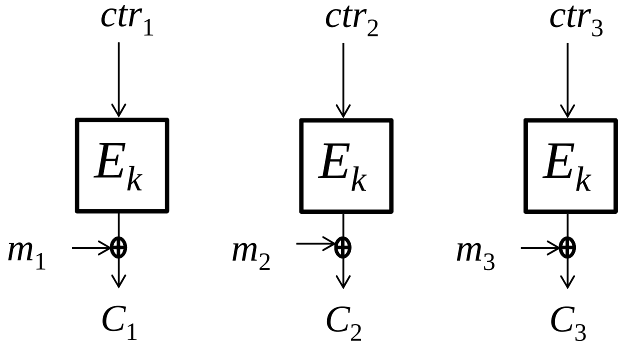


Figure 2.4: Counter (CTR) mode

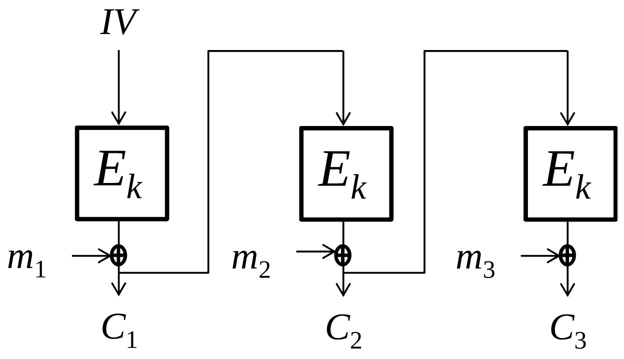


Figure 2.5: Cipher Feedback (CFB) mode

where $k \leftarrow \{0,1\}^n$ is chosen uniformly at random and f is chosen uniformaly at random from the set of permutation on n-bit string.

The definition of a *strong pseudo random permutation* is given as follows.

Definition 4. [**42**] *Let $F : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ be an efficient, keyed permutation. We say that F is a strong pseudo permutation if for all probabilistic polynomial-time distinguisher D, there exist a negligible function* `negl` *such that*

$$\left| Pr[D^{F_k(\cdot),F_k^{-1}(\cdot)}(1^n)] - Pr[D^{f(\cdot),f^{-1}(\cdot)}(1^n)] \right| \leq \texttt{negl}(n)$$

where $k \leftarrow \{0,1\}^n$ is chosen uniformly at random and f is chosen uniformaly at random from the set of permutation on n-bit string.

2.1.2 Goal of Attack

Cryptanalysis is to study attacks for violating security claims of cryptographic algorithms. Due to the Kerchoffs's principle, block ciphers should to be resistant to any attack even if the algorithm is public. Generally, goals of attacks on block ciphers are given as follows.

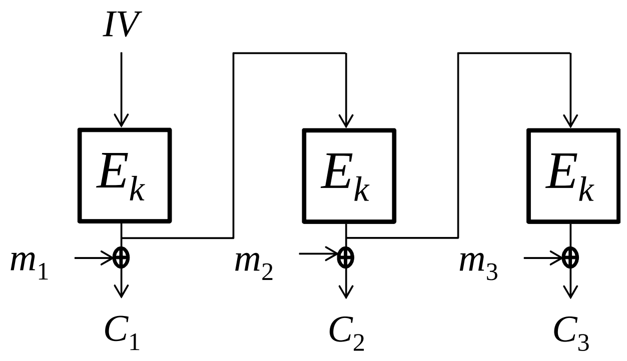


Figure 2.6: Output Feedback (OFB) mode

Key Recovery Attack : Adversary finds a secret key. Once a secret key is found, any ciphertext/plaintext is easily decrypted/encrypted.

Distinguishing Attack : Adversary distinguishes a block cipher from a (strong) random pseudo permutation. This attack violates security requirements of block ciphers as a (strong) pseudo random permutation.

Secure block ciphers should be resistant to distinguishing attacks and key recovery attacks in any reasonable amount of time. If an adversary tries out all possible keys, a secret key can be recovered. This is called a *brute force attack* and surely applicable to any cipher as a generic attack. A block cipher is regarded as secure if the time complexity of best known attack is less than that of the brute force attack. We give the following definition.

Definition 5. *(Successful attack) If the time complexity of an attack is less than that of the brute force attack, such an attack is regarded as a successful attack.*

2.1.3 Attack Scenario

Several types of adversaries are considered in attacks on block ciphers. They know additional information with respect to inputs and outputs of block ciphers. We describe attack scenarios of block ciphers, which are sorted by power of an adversary.

Ciphertext-only attack : Adversary is allowed to obtain only ciphertexts without any plaintext query.

Known plaintext attack : Adversary is allowed to know pairs of a plaintext and the corresponding ciphertext.

Chosen plaintext attack: Adversary is allowed to choose plaintexts to be encrypted and obtain the corresponding ciphertexts.

Chosen ciphertext attack : Adversary is allowed to choose ciphertexts to be decrypted and obtain the corresponding plaintexts.

According to definitions 2 and 3, a pseudo random permutation should be secure (indistinguishable) against a chosen plaintext attack, and a strong random permutation should be secure (indistinguishable) even if a chosen ciphertext attack is allowed.

Following stronger scenarios are also taken into consideration in the line of attacks on block ciphers.

Adaptive plaintext attack : Adversary is allowed to adaptively choose plaintexts, i.e., after obtaining a pair of chosen plaintext and ciphertext, she can send a chosen-plaintext query.

Adaptive ciphertext attack : Adversary is allowed to adaptively choose ciphertexts, i.e., after obtaining a pair of chosen ciphertext and a plaintext, she can send a chosen-ciphertext query.

An adversary aims to violate security claims of block ciphers in these attack scenario. Basically, today's block ciphers should be secure even if an adversary in all scenarios is assumed.

Finally we introduce an other class of an attack scenario called *related-key attacks* [6]. In the related-key scenario, an adversary can encrypt plaintexts/decrypt ciphertexts under multiple unknown keys such that the relation between them is known to the adversary. Though this type of the attack may not lead to a realistic threat in practical security protocols which use the block cipher in a standard way, it is meaningful during the design and certification of ciphers, and also useful for security evaluations of block cipher-based constructions such as message authenticated codes and hash functions.

2.2 Stream Cipher

A stream cipher is a function that converts fixed-length inputs (a key and an Initial Vector (IV)) into an arbitrary-length output called keystream, and provide a data *confidentiality.* A ciphertext/plaintext is obtained by XORing a plaintext/ ciphertext and a keystream. Figure. 2.7 shows a stream cipher of c-bit IV and k-bit key. Formal definition is as follows.

Definition 6. *Stream cipher is a function such that a mapping* $S : \{0,1\}^k \times \{0,1\}^c = \{0,1\}^\ell$*, where* k *is key size,* c *is an IV size and* ℓ *is a keystream size.*

Typically, a stream cipher consists of a key scheduling algorithm (KSA) and a pseudo-random generation algorithm (PRGA). The KSA converts a user-provided key and a IV into a seed, typically called *state*, whose size is more than twice size of a key. The PRGA generates each unit of keystreams with updating the state. Note that a key is kept to be secret, while an IV is a public value.

2.2.1 Security Requirement

Ideally, a stream cipher should behave like a *pseudo random number generator.* The definition of a pseudo random number generator is as follows.

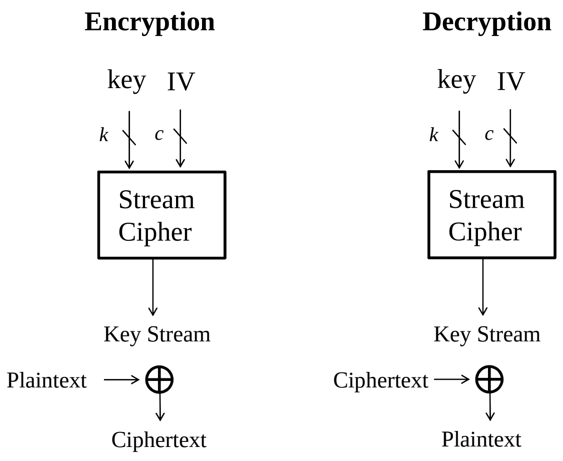


Figure 2.7: Stream cipher of c-bit IV and k-bit key

Definition 7. [42] *Let ℓ be a polynomial and let G be a deterministic polynomial-time algorithm such that for any input $s \in \{0,1\}$, algorithm G outputs a string of length $\ell(n)$. We say that G is a pseudo random generator if following two condition hold:*

1. *(Expansion :) For every n it holds that $\ell(n) \geq n$*
2. *(Pseudorandomness :) For all probabilistic polynomial-time distinguisher D, there exists a negligible function* `negl` *such that :*

$$|Pr[D(r) = 1] - Pr[D(G(s)) = 1]| \leq \texttt{negl}(n),$$

where r is chosen uniformly at random from $\{0,1\}^{\ell(n)}$, the seed s is chosen uniformly at random from $\{0,1\}^n$, and the probabilities are taken over the random coin used by D and the choice of r and s.

2.2.2 Goal of Attack

Goals of attacks on stream ciphers are given as follows.

Key Recovery Attack: Adversary finds a secret key. If a secret key is recovered, any keystream can be obtained. It means that any ciphertext/plaintext is easily decrypted/encrypted.

State Recovery Attack: Adversary recovers a internal state, which is essentially equivalent to the secret key. Once the state is recovered, later units of a keystream can be computed. If the state update function is invertible, all units of a keystream are obtained.

Distinguishing Attack: Adversary distinguishes a keystream from a random stream. It aims to violates a pseudo randomness of an output, which is a security requirement of stream ciphers.

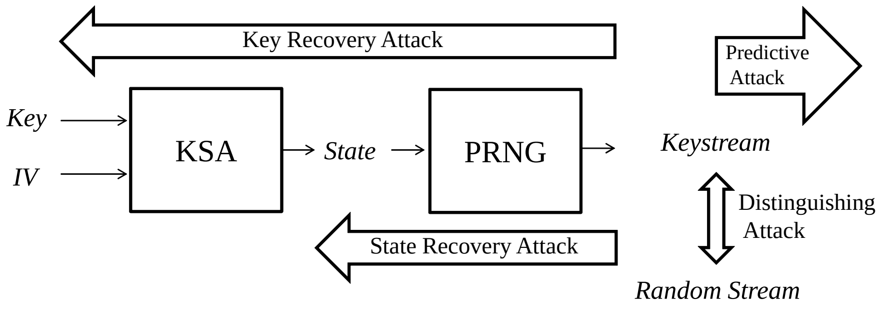


Figure 2.8: Attacks of stream cipher

Prediction Attack: Adversary predicts an unknown keystreams by using the knowledge of known keystreams.

Figure 2.8 shows these attacks. *Secure* stream ciphers should be resistant to all of above attacks in any reasonable amount of time. A stream cipher is regarded as secure if time complexity of best known attacks is less than that of a *brute force search* of the key. In other word, a successful attack of stream ciphers is feasible with less time complexity than the brute force attack.

2.2.3 Attack Scenario

We describe attack scenarios of stream ciphers, which are sorted by power of an adversary.

Ciphertext-only attack : Adversary is allowed to obtain only ciphertexts without any plaintext query.

Known plaintext attack : Adversary is allowed to know pairs of a plaintext and the corresponding ciphertext.

Chosen plaintext attack : Adversary is allowed to choose plaintexts to be encrypted and know the corresponding ciphertexts.

Chosen ciphertext attack : Adversary is allowed to choose ciphertexts to be decrypted and know the corresponding plaintexts.

In these scenarios except a ciphertext-only attack, we can obtain the corresponding keystream by XORing plaintexts and the ciphertexts.

Following scenarios with respect to IV are also considered.

Known IV attack : Adversary is allowed to knows the value of IV.

Chosen IV attack : Adversary is allowed to choose the value of IV.

Note that the combination of these are also considered e.g., a chosen IV known plaintext attack and a know IV chosen ciphertext attack. In general, stream ciphers should be secure even if an adversary in all scenarios is assumed. Similar to block ciphers, a related key attack also is considered in stream ciphers.

2.3 Hash Function

Hash function is a deterministic function that converts an arbitrary-length input (a message) into a fixed-length output (a digest) without using a secret key, and basically provide a data *integrity*. Formally, it is defined as

Definition 8. *Hash function is a mapping* $H : \{0,1\}^m = \{0,1\}^n$, *where* m *is a message size and* n *is digest size.*

2.3.1 Security Requirement

Ideally, a keyed hash function (a family of hash functions) should behave like a pseudo random function. The definition of it is as follows.

Definition 9. [42] *Let* $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *be an efficient, length-preserving, keyed function. We say that* F *is a pseudo permutation if for all probabilistic polynomial-time distinguisher* D*, there exist a negligible function* `negl` *such that*

$$\left| Pr[D^{F_k(\cdot)}(1^n)] - Pr[D^{f(\cdot)}(1^n)] \right| \leq \texttt{negl}(n)$$

where $k \leftarrow \{0,1\}^n$ *is chosen uniformly at random and* f *is chosen uniformly at random from the set of function mapping* n*-bit strings to* n*-bit strings.*

Many cryptographic protocols require a secure hash function which holds several security properties such as classical ones: collision resistance, preimage resistance and second preimage resistance. These are defined as follows.

Collision resistance It should be difficult to find two different messages M_1 and M_2 such that $H(M_1) = H(M_2)$.

Preimage resistance Given a digest d, it should be difficult to find messages M such that $d = H(M)$.

Second preimage resistance Given a pair of a massage M_1 and digest d such that $d = H(M_1)$, it should be difficult to find another messages M_2 such that $d = H(M_2)$.

2.3.2 Goal of Attack

Attacks of hash functions aim to violate security claim such that the collision resistance, the preimage resistance and the second preimage resistance. Unlike block ciphers and stream ciphers, hash functions does not have the secret key.

A hash function is regarded as secure if time complexity of best known attack is less than that of *generic algorithm* of finding collision, preimage and second preimages. When digest size is n, these are given as $2^{n/2}$, 2^n and 2^n, respectively.

2.3.3 Attack Scenario

Hash functions do not use any secret information, and an adversary can freely compute the algorithm of hash functions. We do not consider attack scenarios.

Part I

Analysis of Block Cipher

Chapter 3

A Single-Key Attack on GOST Block Cipher

The GOST block cipher is the Russian encryption standard published in 1989. In spite of considerable cryptanalytic efforts over the past 20 years, a key recovery attack on the full GOST block cipher without any key conditions (*e.g.*, weak keys and related keys) has not been published yet. In this chapter, we show the first single-key attack, which works for all key classes, on the full GOST block cipher. At first, we develop a new attack framework called *Reflection-Meet-in-the-Middle Attack*. This approach combines techniques of the reflection attack and the meet-in-the-middle (MITM) attack. Then, we apply it to the GOST block cipher employing bijective S-boxes. In order to construct the full-round attack, we use additional novel techniques which are the effective MITM techniques using equivalent keys on a small number of rounds. As a result, a key can be recovered with a time complexity of 2^{225} encryptions and 2^{32} known plaintexts. Moreover, we show that our attack is applicable to the full GOST block cipher using any S-boxes, including non-bijective S-boxes.

3.1 Introduction

The GOST block cipher [17] is known as the former Soviet encryption standard GOST 28147-89 which was standardized as the Russian encryption standard in 1989. It is widely used in Russia for commercial and governmental uses, and also adopted as a national standard of surrounding countries such as Belarus, Kazakhstan and Ukraine.

GOST [1] is based on a 32-round Feistel structure with 64-bit block and 256-bit key size. The round function consists of a key addition, eight 4×4-bit S-boxes and a rotation. The GOST standard [17] does not specify a set of S-boxes, and each industry uses a different set of S-boxes given by the government. For example, one set of the S-boxes used in the Central Bank of the Russian Federation is known as in [43]. Although the choice of the S-boxes can also be seen as a part of the secret key, Saarinen has pointed out that a chosen key attack reveals the set of S-boxes with a time complexity of 2^{32} encryptions [44].

[1]For the remainder of this chapter, we refer to the GOST block cipher as GOST.

In addition, GOST is well-suited for compact hardware implementations due to its simple structure. Poschmann *et al.* showed an extremely compact implementation requiring only 651 GE [45]. Therefore, GOST is considered as one of ultra lightweight block ciphers such as PRESENT [46], KATAN family [24], LED [20] and Piccolo [21], which are suitable for the constrained environments including RFID tags and sensor nodes.

Over the past 20 years, a number of attacks on GOST have been published. Kelsey *et al.* first analyzed related-key characteristics of GOST [47]. After that, a differential attack on 13-round GOST was proposed by Seki and Kaneko [48]. In the related-key setting, the attack is improved up to 21 rounds [48]. Ko *et al.* showed a related-key differential attack on the full GOST [49]. These results work only when GOST employs the S-boxes of the Central Bank of the Russian Federation [43]. Fleischmann *et al.* presented a related-key boomerang attack on the full GOST which works for any S-boxes [50]. However, Rudskoy pointed out flaws of this attack and modified it [51]. As another type of attack, Biham *et al.* showed slide attacks on reduced GOST [52]. Their attack utilizes self similarities among round functions of the encryption process, and does not depend on the values of S-boxes. In other words, even if an attacker does not know the S-boxes, 24-round GOST can be attacked by this approach. If the values are known, this attack can be extended up to 30 rounds. In addition, for a class of 2^{128} weak keys, the full GOST can be attacked by this approach. After that, Kara proposed a reflection attack on 30-round GOST [18]. This attack also uses self similarities among round functions, and works for any bijective S-boxes. The difference from the slide attack proposed by Biham *et al.* [52] is to use self similarities between the round function and its inverse function [2]. The reflection attack utilizes these similarities in order to construct fixed points of some round functions. Moreover, for a class of 2^{224} weak keys, the full GOST can be attacked by using the reflection technique.

In spite of considerable cryptanalytic efforts, a key recovery attack on the full GOST without any key assumptions (*e.g.*, weak keys and related keys) has not been published so far. Furthermore, a weak-key attack and a related-key attack are arguable in the practical sense, because of their strong assumptions. A weak-key attack is generally applicable to a small fraction of the keys, *e.g.*, in the attack of [18], the rate of weak keys in all keys is $2^{-32} (= 2^{224}/2^{256})$. Hence, almost all keys, $(2^{256} - 2^{224}) \approx 2^{256}$ keys, cannot be attacked by [18]. Besides, the attacker cannot even know whether a target key is included in a weak key class or not. Only if the key is in the weak key class, the complexity for finding the key is less than that of the exhaustive key search. A related-key attack requires an assumption that the attacker can access the encryption/decryption under multiple unknown keys such that the relation between them is known to the attacker. Though this type of attack is meaningful during the design and certification of ciphers, it does not lead to a realistic threat in practical security protocols which use the block cipher in a standard way as stated in [53] [3]. Therefore, the security under the single-key setting is the most important issue from the aspect of the practical security. In particular,

[2]Another difference between [52] and [18] is rounds to be attacked. [52] proposes an attack on the first 30 rounds while [18] proposes an attack on the last 30 rounds.

[3]In nonstandard uses, there are cases where a related-key weakness leads to realistic threats such as the Microsoft's Xbox attack [54]. In this case, the block cipher TEA [55] is used as a compression function in a hash function.

Table 3.1: Key recovery attacks on GOST

Key setting	Type of attack	Round	Time	Data	Paper
Single key	Differential*1	13	-	2^{51} CP	[48]
	Slide*3	24	2^{63}	2^{63} ACP	[52]
	Slide*3	30	$2^{253.7}$	2^{63} ACP	[52]
	Reflection*2	30	2^{224}	2^{32} KP	[18]
	R-MITM *3	32	2^{225}	2^{32} KP	**Our**
Single key (Weak key)	Slide (2^{128} weak keys) *3	32	2^{63}	2^{63} ACP	[52]
	Reflection (2^{224} weak keys)*2	32	2^{192}	2^{32} CP	[18]
Related key	Differential*1	21	-	2^{56} CP	[48]
	Differential*1	32	2^{244}	2^{35} CP	[49]
	Boomerang *2	32	2^{224}	2^{10} CP	[51]

*1 Works for one specific set of S-boxes in [43]. *2 Works for any bijective S-boxes.
*3 Works for any S-boxes.

an ultra lightweight block cipher does not need a security against related-key attacks in many cases. For example, in low-end devices such as a passive RFID tag, the key may not be changed in its life cycle as mentioned in [46, 24]. Indeed, KTANTAN supports only a fixed key [24] and the compact implementation of GOST proposed by Poschmann *et al.* also uses a hard-wired fixed key [45].

Recently, the Meet-in-the-Middle (MITM) attack on KTANTAN [24] was presented by Bogdanov and Rechberger [9]. Their attack mainly exploits the weakness of the key schedule function of KTANTAN, which is that large parts of the cipher depend on a part of key bits. The MITM attack seems to be effective for block ciphers whose key schedules are simple, *e.g.*, a bit or a word permutation, in the sense that the key dependency is not strong. In fact, the key schedule function of KTANTAN consists of only a bit permutation. Since GOST also has a simple key schedule function, which is a word permutation, the MITM attack seems applicable to it. However, it does not work well on the full GOST, because the key dependency of the full GOST is stronger than that of KTANTAN due to the iterative use of key words in many round functions.

Our Contributions. In this chapter, we first introduce a new attack framework called *Reflection-Meet-in-the-Middle (R-MITM) Attack* ; it is a combination of the reflection attack and the MITM attack. The core idea of this combination is to make use of fixed points of the reflection attack to enhance the MITM attack. If some round functions have fixed points, we can probabilistically remove these rounds from the whole cipher. Since this skip using fixed points allows us to disregard the key bits involved in the removed rounds, the key dependency is consequently weakened. Thus, our attack is applicable to more rounds compared with the original MITM attack if fixed points can be constructed with high probability. Then, we apply it to the GOST block cipher employing bijective S-boxes. Furthermore, to construct the

Table 3.2: Key schedule of GOST

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Key	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8
Round	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Key	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_8	k_7	k_6	k_5	k_4	k_3	k_2	k_1

full-round attack, we use additional novel techniques which are the effective MITM techniques using equivalent keys on a small number of rounds. As a result, we succeed in constructing the first key recovery attack on the full GOST block cipher in the single key setting. It can recover the full key with a time complexity of 2^{225} encryptions, 2^{32} known plaintext/ciphertext pairs and memory of 2^{64} blocks. Moreover, by analyzing the properties of GOST exploited for the attack deeply, we show that this attack can be applied to the full GOST block cipher using any S-boxes, including non-bijective S-boxes, with the same data and time complexities, while the memory depends on the used S-boxes. These results are summarized in Table 1.

3.2 Preliminaries

In this section, we give brief descriptions of GOST, the MITM attack and the reflection attack.

3.2.1 Description of GOST

GOST is a block cipher based on a 32-round Feistel structure with 64-bit block and 256-bit key size.

The 256-bit master key K is divided into eight 32-bit words, $K = (k_1, k_2, \ldots, k_8)$, $k_i \in \{0,1\}^{32}$. These words are used as round keys rk_i $(1 \leq i \leq 32)$ as

$$rk_i = \begin{cases} k_{i-8\times\lfloor\frac{i-1}{8}\rfloor} \ (1 \leq i \leq 24), \\ k_{9-(i-8\times\lfloor\frac{i-1}{8}\rfloor)} \ (25 \leq i \leq 32). \end{cases}$$

See Table 8.2.

Let a 64-bit data state be $L_i||R_i$, $\{L_i, R_i\} \in \{0,1\}^{32}$, where $||$ is a concatenation, $L_0||R_0$ denotes a plaintext P, and $L_{32}||R_{32}$ denotes a ciphertext C. In each round, the state is updated as

$$\begin{aligned} L_{i+1} &= R_i, \\ R_{i+1} &= L_i \oplus (F(rk_i \boxplus R_i)), \end{aligned}$$

where $\boxplus$ is an addition modulo 2^{32} and $\oplus$ is a bitwise exclusive OR. The F-function consists of eight 4×4-bit S-boxes S_j $(1 \leq j \leq 8)$ and an 11-bit left rotation (See Fig.1). Note that the GOST standard [17] does not specify a set of S-boxes, and each industry uses a different set of S-boxes.

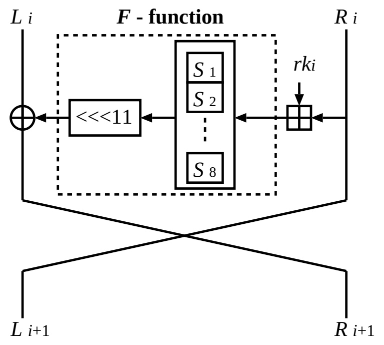


Figure 3.1: One round of the GOST block cipher

3.2.2 Meet-in-the-Middle Attack

The Meet-in-the-Middle (MITM) attack was first proposed by Diffie and Helman [56]; it evaluates the security of the multiple encryptions with distinct keys $key1$ and $key2$ such that $C = E_{key2}(E_{key1}(P))$. The central idea is to compute $E_{key1}(P)$ and $E^{-1}_{key2}(C)$ independently by guessing $key1$ and $key2$. If the guessed key value is correct, the equation $E_{key1}(P) = E^{-1}_{key2}(C)$ holds. Due to the parallel guesses of $key1$ and $key2$, the attack is more effective than an exhaustive search in terms of time complexity. So far, the MITM attack has been applied to several block ciphers [57, 58, 59, 53, 60, 61]. Furthermore, over the past few years, this attack has been improved in a line of preimage attacks on hash functions, and several novel techniques have been introduced, *e.g.*, the splice and cut [16] and the initial structure [62].

The MITM attack consists of two stages: a MITM stage and a key testing stage. First, the MITM stage filters out some wrong key candidates and reduces the key space. Then, the key testing stage finds a correct key from the surviving key candidates in a brute force manner.

Let $E_K : \{0,1\}^b \rightarrow \{0,1\}^b$ be a block cipher with an l-bit key K and a b-bit block. Assume that E_K is a composition of round functions as follows;

$$E_K(x) = F_{k_r} \circ F_{k_{r-1}} \circ \cdots \circ F_{k_1}(x),\ x \in \{0,1\}^b,$$

where r is the number of rounds, $k_1, \ldots, k_r$ are round keys and F_{k_i} is the i-th round function, $F_{k_i} : \{0,1\}^b \rightarrow \{0,1\}^b$. The composition of $j - i + 1$ functions starting from i is denoted by $F_K[i,j]$ defined as

$$F_K[i,j](x) = F_{k_j} \circ \cdots \circ F_{k_i}(x),\ 1 \leq i < j \leq r.$$

In the following, we give details of each stage of the MITM attack.

MITM stage : $E_k(X)$ is divided into two functions as $E_K(X) = F_K[a+1,r] \circ F_K[1,a]$, $1 < a < r - 1$ [4]. Let K_1 and K_2 be sets of key bits used in $F_K[1,a]$ and

[4]As in the attack of KTANTAN [9], by using the partial matching technique, E_K is divided into $F_K[1,a]$ and $F_K[a+t,r]$, $t > 1$. However, we consider only the case of $t = 1$, because we do not use partial matching.

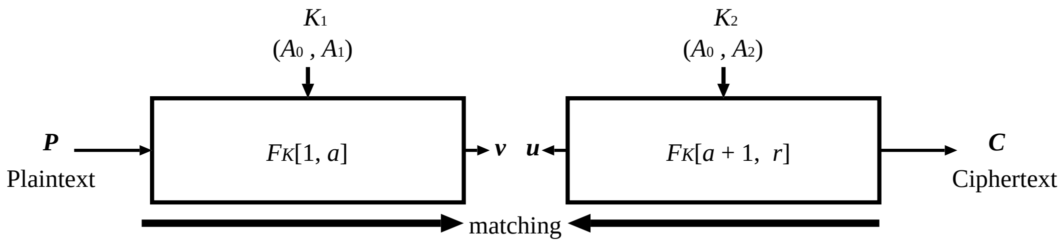


Figure 3.2: Meet-in-the-middle stage

$F_K[a+1,r]$, respectively. $A_0 = K_1 \cap K_2$ is the common set of key bits used in both $F_K[1,a]$ and $F_K[a+1,r]$. $A_1 = K_1 \setminus K_1 \cap K_2$ and $A_2 = K_2 \setminus K_1 \cap K_2$ are the sets of key bits used in only $F_K[1,a]$ and only $F_K[a+1,r]$, respectively. In this stage, we use only one plaintext/ciphertext pair (P, C). [5]

The procedure of the MITM stage is as follows. Fig. 2 shows the overview of the MITM stage.

1. Guess the value of A_0.
2. Compute $v = F_K[1,a](P)$ for all values of A_1 and make a table of (v, A_1) pairs. In this step, $2^{|A_1|}$ pairs are generated, where $|A_i|$ is the bit length of A_i and $2^{|A_i|}$ is the number of elements of A_i.
3. Compute $u = F_K^{-1}[a+1,r](C)$ for all values of A_2. In this step, $2^{|A_2|}$ pairs are generated.
4. Add key candidates for which the equation $v = u$ is satisfied to the list of surviving keys. The number of surviving keys is $2^{|A_1|+|A_2|}/2^b$.
5. Repeat steps 2-4 for each different value of A_0 ($2^{|A_0|}$ times).

In this stage, 2^{l-b} key candidates survive, because $2^{|A_1|+|A_2|}/2^b \times 2^{|A_0|} = 2^l/2^b$.

Key testing stage : We test surviving keys in a brute force manner by using additional plaintext/ciphertext pairs.

We evaluate the cost of this attack. The whole attack complexity C_{comp} is estimated as

$$C_{comp} = \underbrace{2^{|A_0|}(2^{|A_1|} + 2^{|A_2|})}_{\text{MITM stage}} + \underbrace{(2^{l-b} + 2^{l-2b} + \ldots)}_{\text{Key testing stage}}.$$

The number of required plaintext/ciphertext pairs is $\lceil \frac{l}{b} \rceil$. The required memory is $\min(2^{|A_1|}, 2^{|A_2|})$ blocks[6] , which is the cost of the table used in the MITM stage. When $\min(|A_1|, |A_2|) > 1$, the attack is more effective than an exhaustive search. Therefore, the point of the MITM attack is to find independent sets of master key bits such as A_1 and A_2.

[5]Though more such pairs can be used to reduce the key space, this chapter only treats a single pair. This constraint is necessary for utilizing the equivalent-key technique in our attack.

[6]When $|A_1| > |A_2|$, it is possible to swap roles of A_1 and A_2.

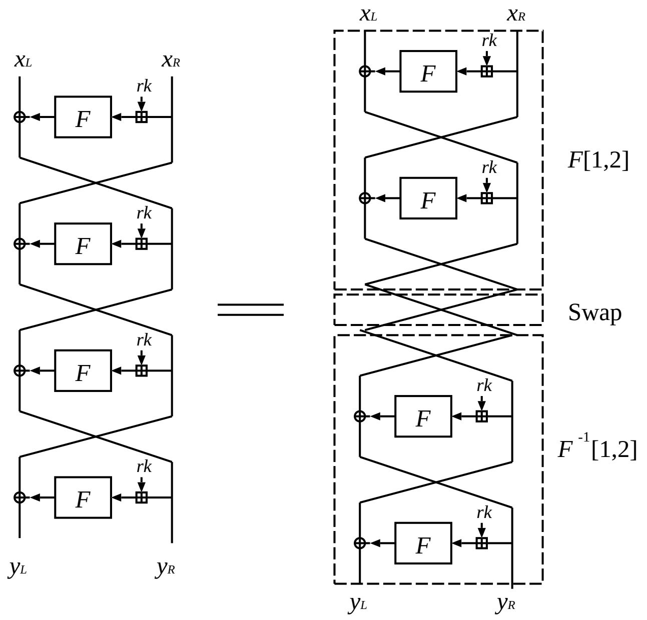


Figure 3.3: Equivalent descriptions of the 4-round Feistel structure with the same round key rk in all round

3.2.3 Reflection Attack

The reflection attack was first introduced by Kara and Manap [63]; it was applied to Blowfish [64]. After that, the attack was generalized by Kara [18]. In this section, we introduce the basic principle of the reflection attack used in our attack. See [18, 63] for more details about the reflection attack.

The reflection attack is a kind of a self-similarity attack such as the slide attack [10, 65]. Though the reflection attack utilizes similarities between the round function and its inverse function, the slide attack exploits similarities among the round functions of only the encryption process. The reflection attack utilizes these similarities in order to construct fixed points of some round functions.

Let $U_K(i,j)$ be the set of fixed points of the function $F_K[i,j]$ defined as follows;

$$U_K(i,j) = \{x \in \{0,1\}^n \mid F_K[i,j](x) = x\}.$$

The basic principle of the reflection attack is given by the following Lemma.

Lemma 1. [18] *Let i and j be integers such that $0 \leq j - i < i + j < r$. Assume that $F_{k_{i-t}} = F^{-1}_{k_{j+t}}$ for all $t : 1 \leq t < i$. If $F_K[i-t, i-1](x) \in U_K(i,j)$, then $x \in U_K(i-t, j+t)$ for all $t : 1 < t < i$. In addition, if $x \in U_K(i-t, j+t)$ for certain $t : 1 < t < i$, then $F_K[i-t, i-1](x) \in U_K(i,j)$.*

For the explanation of Lemma 1, we consider a 4-round Feistel structure with 64-bit block, $E_K : \{0,1\}^{64} \rightarrow \{0,1\}^{64}$. Assume that each round uses the same round key rk, then E_K is expressed as

$$\begin{aligned} E_K &= F_K[3,4] \circ F_K[1,2] \\ &= F_K^{-1}[1,2] \circ S \circ F_K[1,2], \end{aligned}$$

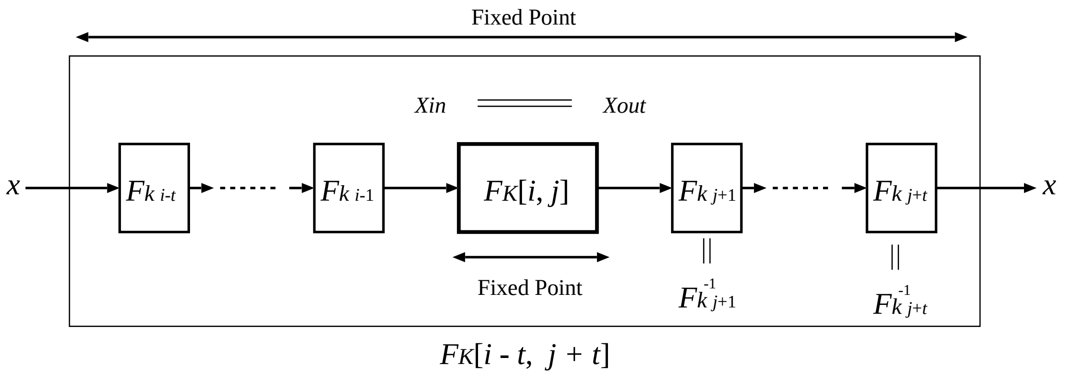


Figure 3.4: Basic principle of the reflection attack

where S is the swap of the Feistel structure. Figure 3 shows these equivalent descriptions of the 4-round Feistel structure. S has 2^{32} fixed points, because the probability of that the right halves equal to the left halves is 2^{-32}. Thus, E_K also has 2^{32} fixed points, *i.e.*, $|U_K(1,4)| = 2^{32}$ due to the property of $F_K[3,4] = F_K^{-1}[1,2]$.

Lemma 1 shows that if the round functions hold the conditions, a local fixed point is expanded to previous and next rounds as shown in Fig. 4. Roughly speaking, such a cipher may have fix points of the long round functions if local fixed points are found. These fixed points enable us to probabilistically skip the round functions in a whole cipher.

We give an example to explain this skip in detail. Let i and j be integers such that $0 < j - i < i + j < r$. Assume that $F_{k_{i-t}} = F^{-1}_{k_{j+t}}$ for all $t : 1 < t < i$, and $E_K(x)$ is expressed as follows;

$$\begin{aligned} E_K(x) &= F_K[j+i,r] \circ F_K[j+1, j+i-1] \circ F_K[i,j] \circ F_K[1, i-1](x) \\ &= F_K[j+i,r] \circ F_K^{-1}[1, i-1] \circ F_K[i,j] \circ F_K[1, i-1](x). \end{aligned}$$

Additionally, assume $F_K[1, i-1](x) \in U_K(i,j)$, then $F_K[1, j+i-1](x) = x$ (Lemma 1). Thus $E_K(x)$ is expressed as

$$E_K(x) = F[j+i, r](x).$$

In this case, the round functions $F_K[1, j+i-1]$ can be skipped from E_K. The probability P_{ref} of that above skip occurs for arbitrary x is $|U_K(i,j)|/2^b$. If $P_{ref} > 2^{-b}$ (*i.e.*, $|U_K[i,j]| > 1$), this skip occurs at $F_K[1, j+i-1]$ with higher probability than a random function.

3.3 Reflection-Meet-in-the-Middle Attack

We propose a new attack framework called *reflection-meet-in-the-middle (R-MITM) attack*, which is a combination of the reflection and the MITM attacks. As mentioned in Section 2.2, the point of the MITM attack is to construct independent sets of master key bits. In general, if the master key bits are used iteratively in each round and the use of key bits is not biased among rounds [7], it seems to be difficult to find the independent sets of master key bits, because such a cipher has a strong key dependency even for a small number of rounds.

To overcome this problem, we utilize the technique of the reflection attack. In the reflection attack, some rounds satisfying certain conditions can be skipped from the whole cipher with probability P_{ref}. From now on, we call this skip a *reflection skip.* Since key bits used in skipped round functions can be omitted, it becomes easier to construct independent sets of master key bits. Thus, the R-MITM attack is applicable to more rounds compared with the original MITM attack when the reflection skip occurs with high probability. This is the concept of the R-MITM attack. In the following, we give a detailed explanation of the attack.

3.3.1 Details of the R-MITM Attack

Suppose that E_K is expressed as follows;

$$E_K(x) = F_K[a_3+1, r] \circ F_K[a_2+1, a_3] \circ F_K[a_1+1, a_2] \circ F_K[1, a_1](x),$$

where $2 < a_1+1 < a_2 < a_3-1 < r-2$ and the reflection skip occurs at $F_K[a_2+1, a_3]$ with probability P_{ref}. Then, E_K can be redescribed as follows and denoted by $E^{'}_K(x)$,

$$E^{'}_K(x) = F_K[a_3+1, r] \circ F_K[a_1+1, a_2] \circ F_K[1, a_1](x).$$

The R-MITM attack consists of three stages; a data collection stage, an R-MITM stage and a key testing stage. In the following, we explain each stage.

Data collection stage : We collect plaintext/ciphertext pairs to obtain a pair in which the reflection skip occurs at $F_K[a_2+1, a_3]$. Since the probability of this event is P_{ref}, the number of required plaintext/ciphertext pairs is P_{ref}^{-1}.

After that, the R-MITM stage and the key testing stage are executed for all plaintext/ciphertext pairs obtained in the data collection stage.

R-MITM stage : We divide E_K into two functions: $F_K[1, a_1]$ and $F_K[a_1+1, r]$[8]. In this stage, we ignore $F_K[a_2+1, a_3]$ as follows;

$$F'_K[a_1+1, r] = F_K[a_3+1, r] \circ F_K[a_1+1, a_2],$$

assuming that the reflection skip occurs. Let K_1 and K_2 be sets of key bits used in $F_K[1, a_1]$ and $F'_K[a_1+1, r]$, respectively. $A_0 = K_1 \cap K_2$ is the set of key bits used in both $F_K[1, a_1]$ and $F'{}_K[a_1+1, r]$. $A_1 = K_1 \setminus K_1 \cap K_2$ and $A_2 = K_2 \setminus K_1 \cap K_2$ are the sets of key bits used in only $F_K[1, a_1]$ and only $F'{}_K[a_1+1, r]$, respectively. Figure 5 illustrates the R-MITM stage.

The procedure of the R-MITM stage is almost the same as the MITM stage of Section 2.2. The difference is that in the R-MITM stage, we assume that a reflection

[7]In KTANTAN [24], 6 bits of master key are not used in the first 111 rounds and another 6 bits of master key are not used in the last 131 rounds. The attack of [9] utilizes this bias of used key bits among rounds.

[8]Though there are many choices of divisions, we use it as an example.

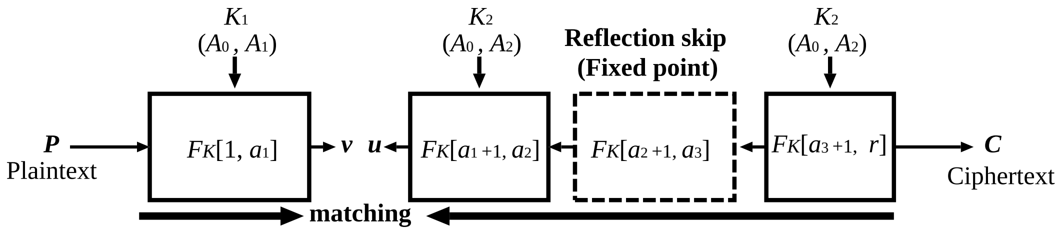


Figure 3.5: Reflection-meet-in-the-middle stage

skip occurs, *i.e.*, $F_K[a_2 + 1, a_3]$ is ignored. After this stage, 2^{l-b} key candidates survive.

Key testing stage : We test surviving keys in a brute force manner by using several plaintext/ciphertext pairs.

Evaluation of the R-MITM Attack

We evaluate the cost of the R-MITM attack. The whole attack complexity C_{comp} is estimated as

$$C_{comp} = (\underbrace{(2^{|A_0|}(2^{|A_1|} + 2^{|A_2|}))}_{\text{R-MITM stage}} + \underbrace{(2^{l-b} + 2^{l-2b} + \ldots)}_{\text{Key testing stage}}) \times P_{ref}^{-1}.$$

The number of required plaintext/ciphertext pairs is $\max(\lceil l/b \rceil, P_{ref}^{-1})$. The required memory is $\min(2^{|A_1|}, 2^{|A_2|})$ blocks, which is the cost of the table in the R-MITM stage. When $\min(2^{|A_1|}, 2^{|A_2|}, 2^b) > (P_{ref}^{-1})$, the attack is more effective than an exhaustive search.

Compared with the basic MITM attack in Section 2.2 for the R-MITM attack, the number of required plaintext/ciphertext pairs increases, because the R-MITM attack utilizes the probabilistic event, *i.e.*, reflection skip. In addition, more independent key bits are needed for the successful attack. However, the R-MITM attack has a distinct advantage, which is the ability to skip some round functions by the reflection skip. Recall that it is the essential for the MITM attack to find independent sets of master key bits. Since the reflection skip enables us to disregard key bits involved in some rounds, it obviously becomes easier to construct such independent sets. Thus, this attack seems to be applicable to more rounds than the MITM attack when the reflection skip occurs with high probability.

3.4 R-MITM Attack on the Full GOST Block Cipher

In this section, we apply the R-MITM attack to the full GOST block cipher [17]. From Table 2, in the whole 32 rounds, the master key is iteratively used four times and all master key bits are used in the each 8-round units, *i.e.*, 1-8, 9-16, 17-24 and 25-32 rounds. The basic MITM attack in Section 2.2 is not applicable to the full GOST, because independent sets of master key bits cannot be constructed in any

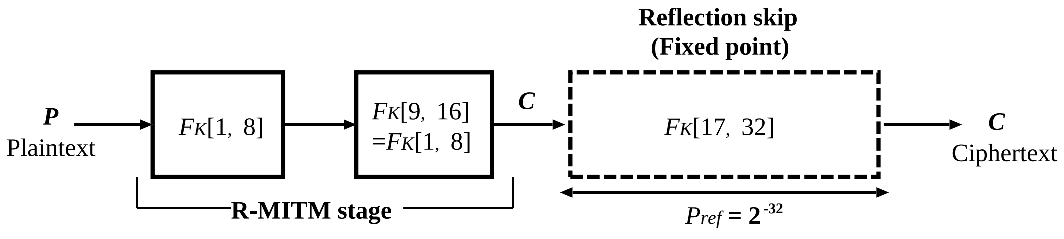


Figure 3.6: Reflection skip of GOST

divisions of 32 rounds. However, by using the R-MITM attack, we can construct independent sets and mount a key recovery attack on the full GOST. In this attack, we do not care about specific values of the S-boxes, and only assume that the S-boxes are bijective.

We first introduce the reflection property of GOST proposed by Kara [18] to construct the reflection skip. Next, we present an effective MITM techniques to enhance the R-MITM stage. These techniques make use of the equivalent keys of four round functions. Finally, we evaluate our attack.

3.4.1 Reflection Property of GOST

The reflection attack on GOST has been proposed by Kara [18] [9]. The GOST block cipher $E_K : \{0,1\}^{64} \to \{0,1\}^{64}$ is expressed as

$$\begin{aligned} E_K &= S \circ F_K[25,32] \circ F_K[17,24] \circ F_K[9,16] \circ F_K[1,8] \\ &= F_K^{-1}[1,8] \circ S \circ F_K[1,8] \circ F_K[1,8] \circ F_K[1,8]. \end{aligned}$$

As mentioned Section 2.3, S has 2^{32} fixed points. From Lemma 1, $F_K^{-1}[1,8] \circ S \circ F_K[1,8]$ also has 2^{32} fixed points, *i.e.*, $|U_K(17,32)| = 2^{32}$. Thus, with probability $P_{ref} = 2^{-32}$ $(= (2^{32}/2^{64}))$, $F_K[17,32]$ can be ignored. E_K is redescribed as follows and denoted by $E_K^{'}$

$$E_K^{'} = F_K[1,8] \circ F_K[1,8].$$

Figure 6 shows this reflection skip of GOST.

Therefore, in the data collection stage, we need to collect $P_{ref}^{-1} = 2^{32}$ plaintext/ciphertext pairs. In 2^{32} collected pairs, there is a pair in which the reflection skip occurs, *i.e.*, last 16 rounds can be removed from E_K.

3.4.2 Effective MITM Technique Using Equivalent Keys on Four Rounds

In the R-MITM stage, we mount the MITM approach on only $E_K^{'} = F_K[1,8] \circ F_K[1,8]$ for all 2^{32} collected pairs.

As described in Section 3.2, we need to construct independent sets A_1 and A_2 which hold the condition, $\min(2^{|A_1|}, 2^{|A_2|}) > 2^{32}$. However, despite the reduction of

[9]The similar technique for constructing a fixed point is also used in the attacks on the GOST hash function [66, 67].

rounds by the reflection skip, in the straightforward method, we cannot find such sets in any divisions of 16 rounds, due to the strict condition of independent sets.

We introduce effective MITM techniques which make use of equivalent keys of four round functions. The aim of these techniques is to ignore the first and the last four rounds and to mount the MITM approach in only intermediate eight rounds. These techniques enable us to construct independent sets enough for the successful attack.

We treat $E_K^{'}$ as the following four-round units;

$$E_K^{'} = F_K[5,8] \circ F_K[1,4] \circ F_K[5,8] \circ F_K[1,4].$$

In the following, we first explain equivalent keys used in our attack. Then, we present detail of the R-MITM stage using the equivalent keys.

3.4.3 Equivalent Keys on Four Rounds.

Define a set of equivalent keys on $F_K[i,j]$ as follows:

$$Z(F_K[i,j], x, y) = \{ek \in \{0,1\}^{256} \mid F_{ek}[i,j](x) = y\},$$

where $(x, y) \in \{0,1\}^{64}$. Note that the class of keys defined above is the equivalent keys with respect to only one input/output pair. To put it more concretely, if equivalent keys $ek \in Z(F_K[i,j], x, y)$ are used, an input x is always transformed to y in $F_K[i,j]$. For other input/output pairs, these relations do not hold even if the same equivalent keys are used.

GOST has an interesting property regarding the equivalent keys on four rounds as described in the following observation.

Observation 1 : *Given any x and y, $Z(F_K[1,4], x, y)$ and $Z(F_K^{-1}[5,8], x, y)$ can be easily obtained, and the number of equivalent keys for each pair (x, y) is* 2^{64}.

For $F_K[1,4]$, k_1, k_2, k_3 and k_4 are added in each round. Given the values of k_1 and k_2, the other values of k_3 and k_4 are determined from $F_K[1,2](x)$ and y as follows:

$$k_3 = F^{-1}(z_L + y_L) - z_R, \tag{3.1}$$

$$k_4 = F^{-1}(z_R + y_R) - y_R, \tag{3.2}$$

where F^{-1} is the inverse of F-function, y_L and y_R are left and right halves of y, and z_L and z_R are those of $F_K[1,2](x)$. Since the values of (k_1, k_2) are 64 bits, the number of $Z(F_K[1,4], x, y)$ is 2^{64}. Figure 7 shows this procedure. A similar property holds for $F_K^{-1}[5,8]$.

From Observation 1, we can easily obtain 2^{64} equivalent keys of the first and the last four rounds for any input/output pairs. Besides, $F_K[1,4]$ and $F_K^{-1}[5,8]$ use different key bits each other, $K_a = (k_1||k_2||k_3||k_4)$ and $K_b = (k_5||k_6||k_7||k_8)$, respectively. Thus, $Z(F_K[1,4], x, y)$ and $Z(F_K^{-1}[5,8], x, y)$ can be expressed by sets of only K_a and K_b as follows;

$$\begin{aligned} Z_{Ka}(F_K[1,4], x, y) &= \{ek_a \in \{0,1\}^{128} \mid F_{ek_a}[1,4](x) = y\}, \\ Z_{Kb}(F_K^{-1}[5,8], x, y) &= \{ek_b \in \{0,1\}^{128} \mid F_{ek_b}^{-1}[5,8](x) = y\}. \end{aligned}$$

Since K_a and K_b are independent sets of mater key, $Z_{Ka}(F_K[1,4], x, y)$ and $Z_{Kb}(F_K^{-1}[5,8], x, y)$ are also independent sets.

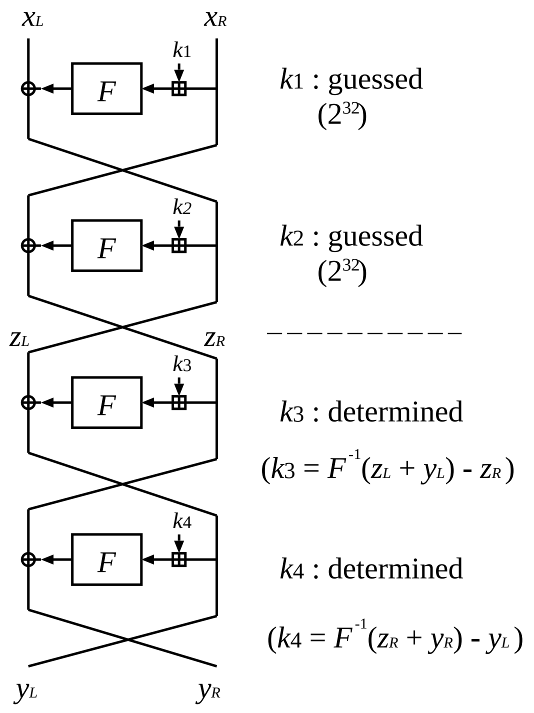


Figure 3.7: Equivalent keys of four rounds

3.4.4 R-MITM Stage Using Equivalent Keys.

Let S and T be $F_K[1,4](P)$ and $F^{-1}[5,8](C)$, which are the input and output values of intermediate eight rounds, *i.e.*, $F_K[5,12] = F_K[1,4] \circ F_K[5,8]$. From Observation 1, given the values of P, C, S and T, we can easily obtain two sets of 2^{64} equivalent keys, $Z_{K_a}(F_K[1,4], P, S)$ and $Z_{K_b}(F_K^{-1}[5,8], C, T)$.

When $Z_{K_a}(F_K[1,4], P, S)$ and $Z_{K_b}(F_K^{-1}[5,8], C, T)$ are used, S and T are not changed. Thus by using these equivalent keys, the first and the last four round can be ignored, and we can mount the MITM attack between $F_K[5,8](S)$ and $F_K^{-1}[1,4](T)$. The number of elements in each independent set is 2^{64}, which is enough for a successful attack.

The procedure of the R-MITM stage is as follows and illustrated in Fig. 8.

1. Guess the values S and T.
2. Compute $v = F_K[5,8](S)$ with 2^{64} K_b in $Z_{K_b}(F_K^{-1}[5,8], C, T)$ and make a table of (v, K_b) pairs.
3. Compute $u = F_K^{-1}[1,4](T)$ with 2^{64} K_a in $Z_{K_a}(F_K[1,4], P, S)$.
4. Add key candidates for which the equation $v = u$ is satisfied to the list of surviving keys. The number of surviving keys is $2^{64+64}/2^{64} = 2^{64}$.
5. Repeat 2-4 with the different values of S and T (2^{128} times).

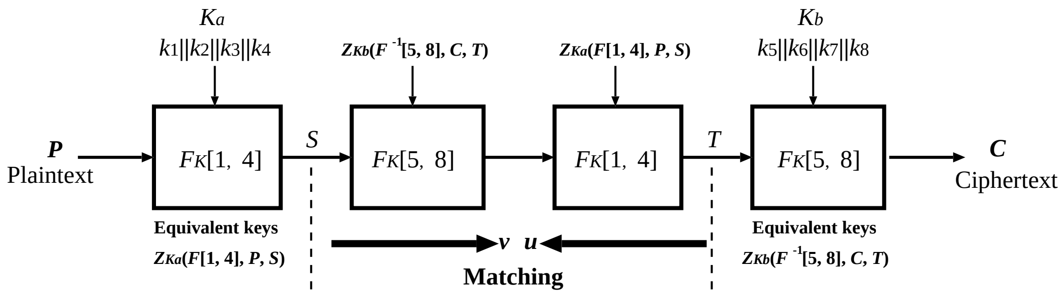


Figure 3.8: R-MITM stage using equivalent keys

After this procedure, 2^{192} (=$2^{64} \times 2^{128}$) key candidates survive. These key candidates are evaluated in the key testing stage.

The R-MITM stage utilizes equivalent-key sets of $Z_{K_a}(F_K[1,4], P, S)$ and $Z_{K_b}(F_K^{-1}[5,8], C, T)$, $0 \leq S,\ T < 2^{64}$, where each set includes 2^{64} elements.

For $Z_{K_a}(F_K[1,4], P, S)$, $0 \leq S < 2^{64}$, from Eq. (1) and (2), all elements of every set are surely different under the assumption that the S-boxes are bijective. Thus, $Z_{K_a}(F_K[1,4], P, S), 0 \leq S < 2^{64}$ covers all 2^{128} ($= 2^{64} \times 2^{64}$) values of K_a. A similar property holds for K_b. Therefore, all possible values for the master key are tested and the set of surviving key candidates surely contain the correct key if the reflection skip occurs.

3.4.5 Evaluation

The whole attack complexity C_{comp} is estimated as

$$\begin{aligned} C_{comp} &= (\underbrace{(2^{128}(2^{64} + 2^{64}))}_{\text{R-MITM stage}} + \underbrace{(2^{256-64} + 2^{256-128} + \ldots)}_{\text{Key testing stage}}) \times 2^{32}, \\ &= 2^{225}. \end{aligned}$$

The number of required known plaintext/ciphertext pairs is $\max(\lceil l/b \rceil, P_{ref}^{-1}) = \max(\lceil 256/64 \rceil, 2^{32}) = 2^{32}$. The required memory is $\min(2^{64}, 2^{64}) = 2^{64}$ blocks, which is the cost of the table used in the R-MITM stage. Therefore, this attack can recover a key with a time complexity of 2^{225} encryptions, 2^{32} known plaintext/ciphertext pairs and 2^{64} memory. It is more effective than an exhaustive attack in terms of time complexity.

3.5 Discussion : Attack without Constraints on S-boxes

In this section, we consider an attack without constraints on the used S-boxes unlike the attack in Section 4 which works only when GOST employs bijective S-boxes. The property of bijective S-boxes is utilized for finding equivalent keys of four rounds (Observation 1). Given a pair $(x,\ y)$, the values of k_3 and k_4 are determined from the values of k_1 and k_2 by using Eq. (1) and (2). Under the assumption that the S-boxes are bijective, since the F-functions are also bijective, each k_3 and k_4 has only one solution with respect to arbitrary values of k_1 and k_2. Thus, the number of equivalent keys for each pair $(x,\ y)$ is 2^{64}, because the joint length of k_1 and k_2 are 64 bits. Thus, the time complexity of the R-MITM stage is easily estimated as $2^{128}(2^{64}+2^{64})$ encryptions.

For an arbitrary S-box, including non-bijective S-boxes, it is difficult to estimate the number of equivalent keys for each pair $(x,\ y)$. When the F-functions is not bijective, it is not guaranteed that each k_3 and k_4 has only one solution for Eq. (1) and (2). In other words, there are cases where k_3 and k_4 have more than one solution or no solution for particular values of k_1 and k_2. Thus, the number of equivalent keys for each pair $(x,\ y)$ strongly depends on the used S-boxes.

However, we are still able to evaluate the whole complexity of the attack. Define the number of equivalent keys in a set $Z(F_K[i,j],x,y)$ as $Z(F_K[i,j],x,y)$. In the R-MITM stage, for all 2^{128} values of S and T, v and u are computed by using sets of equivalent keys $Z_{K_b}(F_K^{-1}[5,8],C,T)$ and $Z_{K_a}(F_K[1,4],P,S)$, respectively. The complexity in the R-MITM stage is estimated as

$$\begin{aligned}
&\sum_{S=0}^{2^{64}-1}\left(\sum_{T=0}^{2^{64}-1}\left(\#Z_{K_b}(F_K^{-1}[5,8],C,T)+\#Z_{K_a}(F_K[1,4],P,S)\right)\right)\\
=\ &2^{64}\cdot\left(\sum_{T=0}^{2^{64}-1}\#Z_{K_b}(F_K^{-1}[5,8],C,T)\right)+2^{64}\cdot\left(\sum_{S=0}^{2^{64}-1}\#Z_{K_a}(F_K[1,4],P,S)\right).
\end{aligned}$$

According to the definition of the equivalent keys in Section 4.2,

$$\sum_{T=0}^{2^{64}-1}\#Z_{K_b}(F_K^{-1}[5,8],C,T)=\sum_{S=0}^{2^{64}-1}\#Z_{K_a}(F_K[1,4],P,S)=2^{128}.$$

Thus, the time complexity of the R-MITM stage is $2^{193}(=2^{64}\cdot 2^{128}+2^{64}\cdot 2^{128})$ encryptions. In addition, the condition for surviving keys is not changed, *i.e.*, $v=u$.

Since the reflection property does not depend on the S-boxes, the number of required known plaintext/ciphertext pairs is also 2^{32}. The whole complexity is estimated as

$$\begin{aligned}
C_{comp}\ &=\ (\underbrace{2^{193}}_{\text{R-MITM stage}}+\underbrace{(2^{256-64}+2^{256-128}+\ldots)}_{\text{Key testing stage}})\times 2^{32}\\
&=\ 2^{225}.
\end{aligned}$$

Therefore, even if arbitrary S-boxes, including non-bijective S-boxes, the complexity and required data are the same as those of the attack on GOST employing bijective S-boxes in Section 4.

However, the required memory size is different, and it depends on the used set of S-boxes. The required memory, which is used for the matching, is estimated as the maximum value of the set $\{\min(\#Z_{K_b}(F_K^{-1}[5,8],C,T),\#Z_{K_a}(F_K[1,4],P,S))\mid 0\leq S,\ T<2^{64}\}$. The best case is when:

$$\forall S,T\in\{0,1\}^{64},\ \#Z_{K_b}(F_K^{-1}[5,8],C,T)=\#Z_{K_a}(F_K[1,4],P,S)=2^{64},$$

which holds if and only if all S-boxes are bijective. Then, the required memory is 2^{64} blocks. The worst case is when:

$$\exists S,T,\in\{0,1\}^{64},\ \#Z_{K_b}(F_K^{-1}[5,8],C,T)=\#Z_{K_a}(F_K[1,4],P,S)=2^{128}.$$

Then, the required memory is 2^{128} blocks.

Therefore, although the required memory depends on the values of the used S-boxes, our attack is applicable to the full GOST using any S-boxes with the same data and time complexities.

3.6 Conclusion

This chapter has presented the first single-key attack on the full GOST block cipher without relying on weak key classes. To build the attack, we introduced a new attack framework called *Reflection-Meet-in-the-Middle Attack*, which is the combination of the reflection and the MITM attacks. We then applied it to the full GOST block cipher employing bijective S-boxes. Furthermore, in order to construct the full-round attack, we utilize further novel techniques which make use of equivalent keys of four round functions. These techniques enable us to mount the effective MITM approach. As a result, we succeeded in constructing the first key recovery attack on the full GOST without any key conditions, which works for any bijective S-boxes. Moreover, by analyzing procedures of the attack, we showed that this attack can be extended to GOST employing any S-boxes, including non-bijective S-boxes. Our result shows that GOST does not have the 256-bit security for all key classes, even if a fixed key is used such as in [45].

The idea of the R-MITM attack seems applicable to other block ciphers in which the fixed point can be constructed with high probability and its key schedule is simple in the sense that the key dependency is not strong. Furthermore, the basic principle of the attack does not require the reflection property and fixed points. Other non-random properties of round functions may also be able to be utilized as the skip techniques, *e.g.*, the strong correlations among round functions.

Chapter 4

Security Analysis of Lightweight Block Ciphers XTEA, LED and Piccolo

In this chapter, we investigate the security of the lightweight block ciphers against the meet-in-the-middle (MITM) attack. Since the MITM attack mainly exploits low key-dependency in a key expanding function, the block ciphers having a simple key expanding function are likely to be vulnerable to the MITM attack. On the other hand, such a simple key expanding function leads compact implementation, and thus is utilized in several lightweight block ciphers. However, the security of such lightweight block ciphers against the MITM attack has not been studied well so far. We apply the MITM attack to the ciphers, then give more accurate security analysis for them. Specifically, combining thorough analysis with new techniques, we present the MITM attacks on 29, 8, 16, 14 and 21 rounds of XTEA, LED-64, LED-128, Piccolo-80 and Piccolo-128, respectively. Consequently, it is demonstrated that the MITM attack is the most powerful attack in the single-key setting on those ciphers with respect to the number of attacked rounds. Moreover, we consider the possibility of applying the recent speed-up keysearch based on MITM attack to those ciphers.

4.1 Introduction

Over past years the demand for security in low resource devices such as RFID tags and sensor nodes has been dramatically increased. This motivates cryptographers to design a new cryptographic primitives aimed to such constrained devices, and thus several lightweight block ciphers have been proposed such as PRESENT [46], LED [20] and Piccolo [21]. In such ciphers, several newly developed design tricks are utilized in order to reduce the required gates. For example, a serially computable MDS matrix used in LED provides good diffusion but requires small gate areas. The key expanding function of KTANTAN [24] does not need to update the user provided secret key itself to generate sub-keys. This allows us to significantly reduce the required gates especially in the fixed-key mode, i.e., a secret key is hard-wired. This technique is utilized in several block ciphers such as XTEA [19], LED and Piccolo. We refer such design trick for a key expanding function as a direct-key expansion throughout this chapter. More precisely, the direct-key expansion referred in this chapter is defined that each subkey k_i can be represented by a secret key K and a round constant c_i as $k_i = KP_i(K) \oplus c_i$ or $k_i = KP_i(K) \boxplus c_i$, where KP_i denotes an arbitrary bit permutation.

While the direct-key expansion leads compact implementation peculiarly in the fixed-key mode, it sometimes causes the security problems due to its slow diffusion regarding sub-keys. For instance, it has been known that KASUMI [68], which consists of the direct-key expansion, is vulnerable to the related-key attacks [69]. Though the practical impact on a related-key attack depends on an application and required conditions for related-keys, the security for most of the recently proposed block ciphers having the direct-key expansion against the related-key attack is well analyzed by the designers such as KTANTAN, LED and Piccolo. Therefore these ciphers are expected to be secure against the related-key attack. In fact, for a well-designed direct-key expansion, it is relatively simple to show that there is no unexpected flaw regarding the related-key attack in the key expanding, in contrast to a complicated key expanding such as AES [1]. It is considered as one of the desirable design features of the direct-key expansion.

Meet-in-the-middle (MITM) attack was first introduced in [56]. Since the MITM attack usually exploits the slowness of the diffusion in a key expanding function, the ciphers having the direct-key expansion are likely to be vulnerable to it. In fact, KTANTAN, which is believed to be secure against the related-key attack, has been theoretically broken by the MITM attack [9]. However, the security of the other block ciphers consisting of the direct-key expansion against the MITM attack have not been well studied. Thus, while the direct-key expansion has several desirable features especially in implementation, it is less reliable compared to a complicated key expanding function due to lack of the thorough security analysis.

In this chapter, we reconsider the security of lightweight block ciphers against the MITM attacks. Our target ciphers are XTEA, LED and Piccolo, which have the direct-key expansion. Combining recent advanced techniques with new techniques such as an equivalent transformation technique for SPN ciphers, we succeed in improving the numbers of attacked rounds for those ciphers by the MITM attack. Specifically, we present the MITM attacks on 29 (out of 64), 8 (out of 32), 16 (out of 48), 14 (out of 25) and 21 (out of 31) rounds of XTEA, LED-64, LED-128, Piccolo-80 and Piccolo-128, respectively. Note that all of our attacks are the best with respect to the number of attacked rounds[1] in the single-key setting in literature. This implies that the MITM attack is actually the most effective attack for block ciphers having the direct-key expansion, and its security analysis is necessary. Moreover, we consider the possibility of applying the recent speed-up keysearch based on the MITM attack, which are applied to the full AES [70] and the full IDEA [71], to our target ciphers.

4.2 Target Ciphers

This section gives brief descriptions of our target ciphers including XTEA [19], LED [20] and Piccolo [21]. Note that all of our target ciphers consist of the direct-

[1]The MITM attacks on the 14- and 21-round Piccolo-80 and Piccolo-128 were introduced in [21]. However, the details of those attacks have not been published.

key expansion. In the descriptions, for each algorithm, we use the same notations used in their original papers.

4.2.1 Description of XTEA

XTEA is a 64-round Feistel cipher with 64-bit block and 128-bit key proposed by Needham and Wheeler in 1997 [19]. Let 64-bit state of the round i be $S^i = L^i|R^i$, where $L^i, R^i \in \{0,1\}^{32}$. Each round updates a state by using a 32-bit round key RK^i as

$$L^{i+1} = R^i, \quad R^{i+1} = L^i \boxplus_{(32)} ((\delta^i \boxplus_{(32)} RK^i) \oplus F(R^i)),$$

where $\boxplus_{(32)}$ denotes an addition modulo 2^{32} and δ^i represents the i-th round constant. The function F is defined as $F(x) = ((x \ll 4) \oplus (x \gg 5)) \boxplus_{(32)} x$. Each round key RK^i is derived from a secret key $K = K_0|K_1|K_2|K_3$, $K_i \in \{0,1\}^{32}$ according to the predefined rule (see Table 4.1).

4.2.2 Description of LED

LED is an SPN-type block cipher supporting a 64-bit block and 64-bit to 128-bit keys proposed by Guo et al. in 2011 [20]. For the 64-bit key mode called LED-64, a 64-bit secret key K is XORed to the 64-bit state every 4 rounds. For the 128-bit key mode called LED-128, a 128-bit secret key K is divided into two 64-bit sub-keys K_1 and K_2, then K_1 and K_2 are alternatively XORed to the 64-bit state every 4 rounds. Each round function consists of `AddConstants`, `SubCells`, `ShiftRows` and `MixColumnsSerial` similar to the round function of AES. The numbers of rounds for LED-64 and LED-128 are 32 and 48, respectively.

4.2.3 Description of Piccolo

Piccolo is a 64-bit block cipher supporting 80 and 128-bit keys proposed by Shibutani et al. in 2011 [21]. Piccolo-80 and Piccolo-128 consist of 25 and 31 rounds of a variant of a generalized Feistel network, respectively. Let 64-bit input of the i-th round be $S^i = X_0^i|X_1^i|X_2^i|X_3^i$, $X_j^i \in \{0,1\}^{16}$. Then the $(i+1)$-th round input S^{i+1} is derived as follows:

$$S^{i+1} = RP(X_0^i|(X_1^i \oplus F(X_0^i) \oplus rk_{2i-2})|X_2^i|(X_3^i \oplus F(X_2^i) \oplus rk_{2i-1})),$$

where F is a 16-bit F-function, RP is a 64-bit permutation, and rk_{2i-2} and rk_{2i-1} are round keys introduced into the i-th round. Each 16-bit round key rk_i is derived from an 80-bit secret key $K = k_0|k_1|...|k_4$ or a 128-bit secret key $K = k_0|k_1|...|k_7$, where $k_j \in \{0,1\}^{16}$ according to the predefined manner (see Table 4.2 and 4.3).

4.3 Meet-in-the-Middle Attacks

In this section, we briefly review the MITM attack presented in [9] and several advanced techniques introduced in [72, 16, 62, 70, 71].

The MITM attack consists of the MITM stage and the key testing stage. The attacker first filters out some wrong keys and reduces the key space in the MITM

Table 4.1: Key expanding functions of XTEA,

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
j of K_j for RK^i	0	3	1	2	2	1	3	0	0	0	1	3	2	2	3	1
Round	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
j of K_j for RK^i	0	0	1	0	2	3	3	2	0	1	1	1	2	0	3	3
Round	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
j of K_j for RK^i	0	0	1	0	2	3	3	2	0	1	1	1	2	0	3	3
Round	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
j of K_j for RK^i	0	0	1	3	2	2	3	2	0	1	1	0	2	3	3	2

Table 4.2: Key expanding functions of Piccolo-80

Round i	1		2		3		4		5		6		7		8	
j of K_j for RK^i	2	3	0	1	2	3	4	4	0	1	2	3	0	1	2	3
Round i	9		10		11		12		13		14		15		16	
j of K_j for RK^i	4	4	0	1	2	3	0	1	2	3	4	4	0	1	2	3
Round i	17		18		19		20		21		22		23		24	
j of K_j for RK^i	0	1	2	3	4	4	0	1	2	3	0	1	2	3	4	4
Round i	25															
j of K_j for RK^i	0	1														

stage, then exhaustively searches a correct key from the surviving key candidates in the key testing stage.

The MITM stage first divides an n-bit block cipher E with an ℓ-bit secret key K into two functions $\mathcal{F}_{(1)}$ and $\mathcal{F}_{(2)}$. K is grouped into three sets $\mathcal{K}_{(1)}$, $\mathcal{K}_{(2)}$ and $\mathcal{K}_{(3)}$, where $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are used only in $\mathcal{F}_{(1)}$ and $\mathcal{F}_{(2)}$, respectively, and $\mathcal{K}_{(3)}$ denotes the other bits of K. Such $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are called *neutral key bits* of $\mathcal{F}_{(2)}$ and $\mathcal{F}_{(1)}$, respectively. Then, we can compute $\mathcal{F}_{(1)}(P)$ and $\mathcal{F}^{-1}_{(2)}(C)$ independently by guessing each neutral key bit. If the guessed key value is correct, the equation $\mathcal{F}_{(1)}(P) = \mathcal{F}^{-1}_{(2)}(C)$ holds. Due to parallel guesses of $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$, we can efficiently check whether the guessed key is the correct one. After this stage, we have $2^{\ell-n}(= 2^{|\mathcal{K}_{(1)}|+|\mathcal{K}_{(2)}|}/2^n \times 2^{|\mathcal{K}_{(3)}|})$ key candidates[2]. In the key testing stage, the attacker exhaustively searches a correct key from the surviving key candidates by using additional plaintext/ciphertext pairs. The required computations of the attack in total C_{comp} is estimated as

$$C_{comp} = \underbrace{2^{|\mathcal{K}_{(3)}|}(2^{|\mathcal{K}_{(1)}|} + 2^{|\mathcal{K}_{(2)}|})}_{\text{MITM stage}} + \underbrace{(2^{\ell-n} + 2^{\ell-2n} + 2^{\ell-3n} + ...)}_{\text{Key testing stage}}.$$

The number of the required plaintext/ciphertext pairs is $\lceil \ell/n \rceil$. The required memory is $\min(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|})$ blocks, which is the cost of the table used in the MITM stage.

Here, we review several advanced techniques that enhance the MITM attack.

[2]The number of key candidates can be reduced by repeatedly performing the MITM stage with additional plaintext/ciphertext pairs.

Table 4.3: Key expanding functions of Piccolo-128

Round i	1		2		3		4		5		6		7		8	
j of K_j for RK^i	2	3	4	5	6	7	2	1	6	7	0	3	4	5	6	1
Round i	9		10		11		12		13		14		15		16	
j of K_j for RK^i	4	5	2	7	0	3	4	1	0	3	6	5	2	7	0	1
Round i	17		18		19		20		21		22		23		24	
j of K_j for RK^i	2	7	4	3	6	5	2	1	6	5	0	7	4	3	6	1
Round i	25		26		27		28		29		30		31			
j of K_j for RK^i	4	3	2	5	0	7	4	1	0	7	6	3	2	5		

- **Partial Matching [16] :** When checking $\mathcal{F}_{(1)}(P) = \mathcal{F}_{(2)}^{-1}(C)$, it is not necessary to match all values of the state. By using only part of the state value for the matching, some key bits around the matching point can be omitted.
- **Splice and Cut [72, 16] :** It regards the last and the first rounds of the block cipher as consecutive rounds assuming the chosen plaintext attack or the chosen ciphertext attack. This allows the attacker to freely choose two functions $\mathcal{F}_{(1)}$ and $\mathcal{F}_{(2)}$, though it generally requires more plaintext/ciphertext pairs than the attack without the splice and cut.
- **Initial Structure [62, 70] :** It is used to skip some rounds around the starting point of the MITM attack, by exchanging two neutral key bits in those skipped round. The formal concept of the initial structure is called *biclique* [73, 70].

Since the MITM attack exploits low key-dependency that large parts of the cipher are independent from some key bits, it is considered as one of the most powerful attacks for the direct-key expansion. Moreover, since the MITM attack does not require related-keys, i.e., it works in the single-key setting, its security analysis is considered to be more important than related-key attacks. These are the reasons why we focus on the MITM attack in this work.

4.4 Meet-in-the-Middle Attacks on Lightweight Block Ciphers

In this section, we apply the MITM attack to lightweight block ciphers XTEA, LED and Piccolo. The results on our MITM attacks and known single-key attacks are summarized in Table 10.1.

In the followings, $F[i, j]$ denotes the $(j - i + 1)$ consecutive rounds starting from the i-th round, e.g., $F[3, 6]$ represents 4 consecutive rounds starting from the 3rd round. $X_{i[j]}$ denotes the j-th bit of X_i, where $X_{i[0]}$ is the most significant bit. Also, $X_{i[0-4]}$ denotes the bits $X_{i[0]}$ to $X_{i[4]}$.

4.4.1 Security of XTEA against MITM Attack

The MITM attack on the 23-round reduced XTEA was proposed in [74]. In order to improve this attack, we develop the 12-round partial matching by thoroughly

Table 4.4: Summary of the attacks in the single-key setting

algorithm	#rounds	attack	#attacked rounds	time	data	memory [bytes]	reference
XTEA	64	MITM	23	2^{117}	18 KP	negligible	[74]
		IDA	23	$2^{105.6}$	2^{63} CP	2^{104}	[75]
		MITM	**29**	2^{124}	2^{45} CP	2^{4}	Our
LED-64	32	diff./linear	8^{*1}	-	-	-	[20]
		MITM	**8**	2^{56}	2^{8} CP	2^{11}	Our
LED-128	48	diff./linear	8^{*1}	-	-	-	[20]
		MITM	**16**	2^{112}	2^{16} CP	2^{19}	Our
Piccolo-80	25	MITM	14	-	-	-	[21]
		MITM	**14**	2^{73}	2^{64} CP	2^{8}	Our
Piccolo-128	31	MITM	21	-	-	-	[21]
		MITM	**21**	2^{121}	2^{64} CP	2^{9}	Our

IDA: impossible differential attack, KP: known plaintext, CP: chosen plaintext
$*1$: there is no useful differentials or linear hulls over 8 rounds.

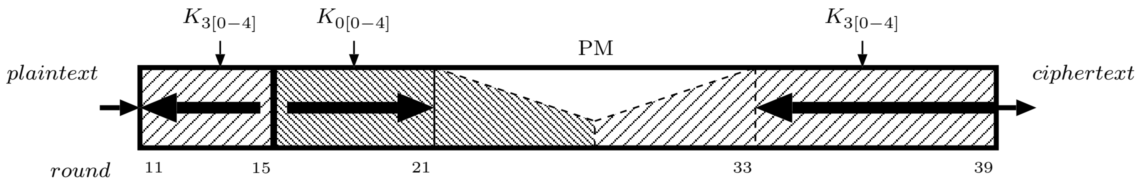


Figure 4.1: Overview of the 29-round attack on XTEA

analyzing the diffusion property of the round functions of XTEA, especially the addition property regarding key differences. Moreover, we carefully choose the value of the starting point in each inner loop to make the splice and cut technique more effective. It enables us to save the amount of the required data in spite of the use of the splice and cut technique. By combining these techniques, we succeed in constructing a MITM attack on the 29-round reduced XTEA. Figure 4.1 shows an overview of the attack. In the figure, PM and IS denote *partial matching* and *initial structure*, respectively.

MITM Attack on 29-round XTEA.

For the 29-round variant of XTEA starting from the round 11, using neutral key bits $\mathcal{K}_{(1)}(=K_{0[0-4]})$ and $\mathcal{K}_{(2)}$ $(=K_{3[0-4]})$, two chunks, $F[16,21]$ and $F^{-1}[34,39] \circ F^{-1}[11,15]$, can be computed independently. The detailed explanations of the partial matching for this attack and the attack complexity are given as follows.

- **Partial Matching.** Due to the differential property of the addition, we can compute $L^{27}_{[30-31]}$ and $R^{27}_{[30-31]}$ from S^{21} without using the value of $K_{3[0-4]}$ in the forward process. On the other hand, $L^{27}_{[30-31]}$ and $R^{27}_{[30-31]}$ are calculated from S^{33} without using the value of $K_{0[0-4]}$ in the backward process as well. Thus, the 12-round partial matching works (see Fig. 4.2).

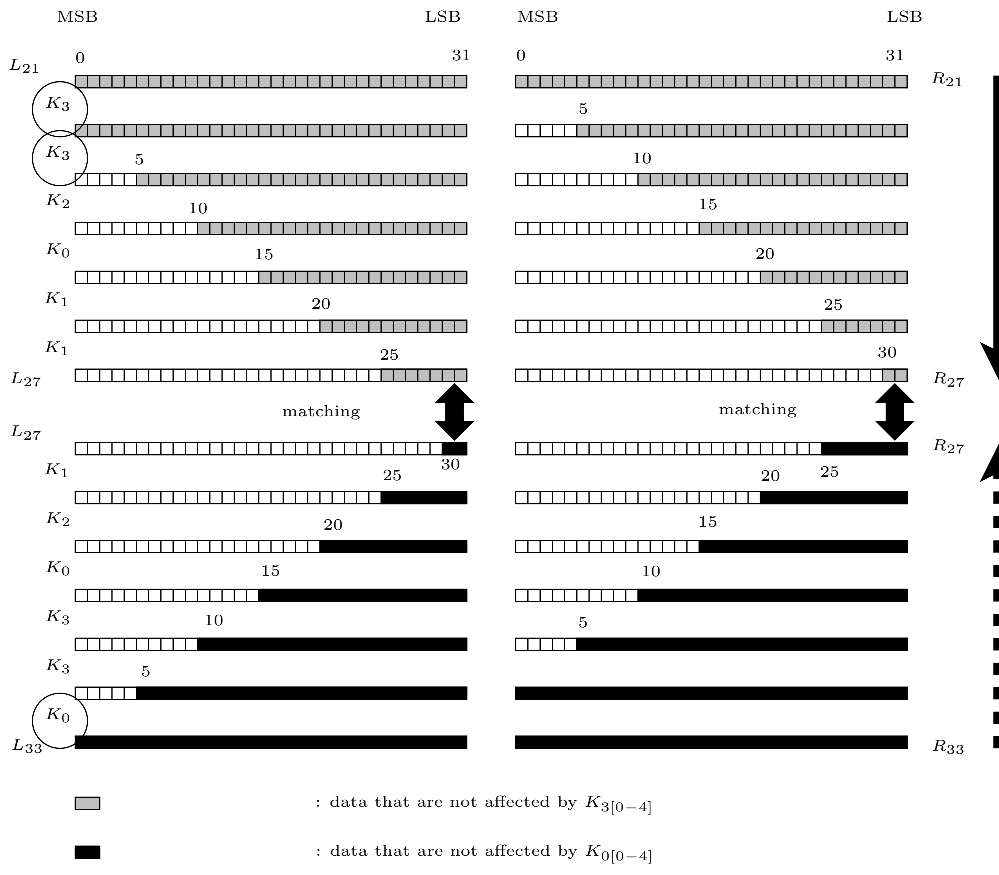


Figure 4.2: 12-round partial matching of XTEA

- **Evaluation.** The time complexity C_{comp} is estimated as

$$C_{comp} = \underbrace{2^{118}(2^5 + 2^5)}_{\text{MITM}} + \underbrace{2^{124}}_{\text{Key testing}} \approx 2^{124}.$$

  The required memory is 2^4 byte ($= 2^5 \cdot 4$ bits). The number of the required plaintext/ciphertext pairs depends on the range of the output of $F^{-1}[11, 15]$. As mentioned in [76], the entire code book seems to be necessary, because $F^{-1}[11, 15]$ contains more key bits than the block size. However, by using the fact that neutral key bits $K_{3[0-4]}$ do not affect $L^{11}_{[25-31]}$ and $R^{11}_{[20-31]}$ as shown in Fig. 4.3, we can fix $L^{11}_{[25-31]}$ and $R^{11}_{[20-31]}$ in the MITM stage. To put it more precisely, we choose the start value as $S_{16} = F[11, 15](S_{11})$ in each value of $\mathcal{K}_{(3)}$, where $L^{11}_{[25-31]}$ and $R^{11}_{[20-31]}$ are set as arbitrary constant, and the other bits are free. Then, during the MITM stage, $L^{11}_{[25-31]}$ and $R^{11}_{[20-31]}$ are always fixed. Thus, the required data is estimated as $2^{45} (= 2^{64-19})$ chosen plaintext/ciphertext pairs.

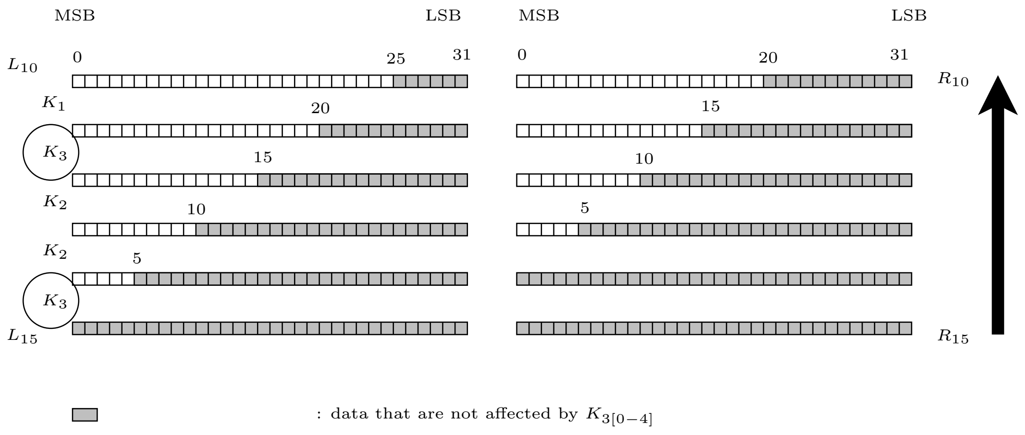


Figure 4.3: $F^{-1}[11, 15]$ of XTEA

4.4.2 Security of LED against MITM Attack

For SPN structures with the direct-key expansion, it seems hard to apply a MITM attack, since all of secret key bits are used in the small number of rounds. In fact, all of secret key bits of LED-64 are introduced in every 4 rounds. However, using *equivalent transformation technique*, we show MITM attacks on the 16-round and the 8-round reduced LED-128 and LED-64, respectively. Figures 4.4 and 4.5 show attack overviews for LED-128 and LED-64, respectively.

MITM Attack on 16-round LED-128.

We consider the 16-round variant of LED-128 starting from the 1st round (i.e., $F[1, 16]$ of LED-128). Let FF and FB be 8 and 4 consecutive round functions starting from the round 1 and 13, respectively, i.e., $FF = F[1, 8]$ and $FB = F[13, 16]$ of LED-128. Also, FB contains both the final and the initial key additions by K_1 as shown in Fig. 4.4. Using neutral key bits K_1 and K_2, two chunks FB and FF can be computed independently. The partial matching used in this attack and the estimated attack complexity are explained as follows.

- **Partial Matching.** The 4-round partial matching is done at rounds 9 to 12 including the key additions before the 9-th round by K_1 and after the 12th round by K_2. $T_1, ..., T_{19}$ denote each state in this part as shown in Fig. 4.6, where each state consists of the 16 4-bit words arranged in a 4×4 square array. $T_i[j]$ denotes the j-th 4-bit word of T_i and $T_i[j, k]$ denotes $T_i[j]$ and $T_i[k]$, e,g., $T_i[3, 7, 11, 15]$ is four 4-bit words in the right most column of T_i. K_1 and K_2 are denoted in a similar way.

  In the straightforward method, it seems hard to construct the 4-round partial matching, since the 2 rounds of LED achieve the full diffusion. We introduce an *equivalent transformation technique* regarding the key addition by K_2. Since MC is a linear operation, T_{19} can be expressed as $T_{19} = MC(T_{17}) \oplus K_2 = MC(T_{17} \oplus K'_2)$, where MC is an operation of `MixColumnsSerial` and $K'_2 = MC^{-1}(K_2)$. This equation means that the operation of `MixColumnsSerial`

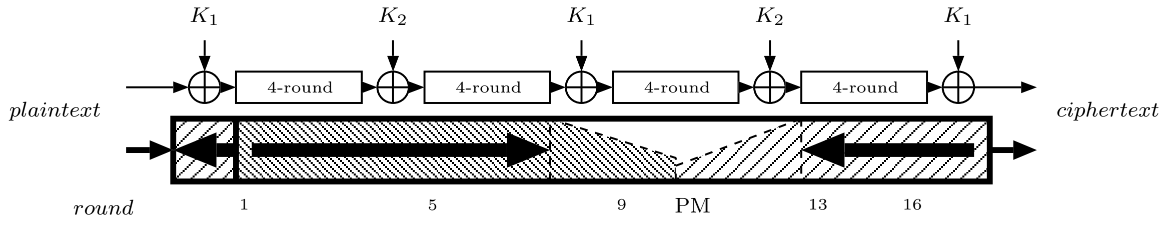


Figure 4.4: Overview of the 16-round attack on LED-128

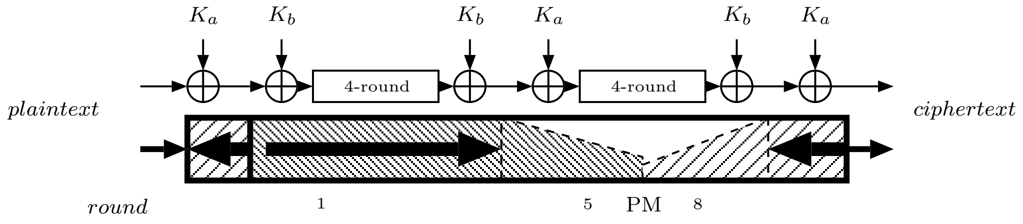


Figure 4.5: Overview of the 8-round attack on LED-64

and the key addition of K_2 are exchangeable by exploiting the linearity of `MixColumnsSerial`. In that case, the diffusion effect of the `MixColumnsSerial` regarding K_2 can be omitted in the backward computation. Since MC^{-1} is an invertible function, even if we directly control the value of K'_2 instead of K_2, the MITM attack surely works in a similar manner.

Besides, we utilize the match through MixColumns technique [77]. Suppose that $\mathcal{K}_{(1)}$ is $K'_2[0,5,10,15]$ and $\mathcal{K}_{(2)}$ is $K_1[0,7,10,13]$, then we can compute the values of $T_9[1,2,3,4,6,7,8,9,11,12,13,14]$ without using $\mathcal{K}_{(2)}$ in the forward computation, and the values of $T_{10}[1,2,3,4,5,6,8,9,11,12,14,15]$ without using $\mathcal{K}_{(1)}$ in the backward computation as shown in Fig. 4.6.

Then we do the matching between these values by using the match through `MixColumnsSerial`. In particular, we obtain the following equations with respect to the first columns of T_{10},

$$\begin{aligned} T_{10}[4] &= (8\cdot T_9[0])\oplus(6\cdot T_9[4])\oplus(5\cdot T_9[8])\oplus(6\cdot T_9[12]), && (4.1)\\ T_{10}[8] &= (B\cdot T_9[0])\oplus(E\cdot T_9[4])\oplus(A\cdot T_9[8])\oplus(9\cdot T_9[12]), && (4.2)\\ T_{10}[12] &= (2\cdot T_9[0])\oplus(2\cdot T_9[4])\oplus(F\cdot T_9[8])\oplus(B\cdot T_9[12]). && (4.3)\end{aligned}$$

For Eqs. (4.1)-(4.3), only $T_9[0]$ is unknown values in the matching. We eliminate this values by combining with these equations. Then we obtain the two independent equations that do not include the variable $T_9[0]$. As an example, we give the following equation obtained from Eqs. (4.1) and (4.2),

$$\begin{aligned} B\cdot T_{10}[4]\oplus 8\cdot T_{10}[8] &= B\cdot((6\cdot T_9[4])\oplus(5\cdot T_9[8])\oplus(6\cdot T_9[12]))\\ &\quad \oplus 8\cdot((E\cdot T_9[4])\oplus(A\cdot T_9[8])\oplus(9\cdot T_9[12])).\end{aligned}$$

Since each column has such two independent 4-bit equations, it is equivalent to 32 $(= 4\times 2\times 4)$-bit matching.

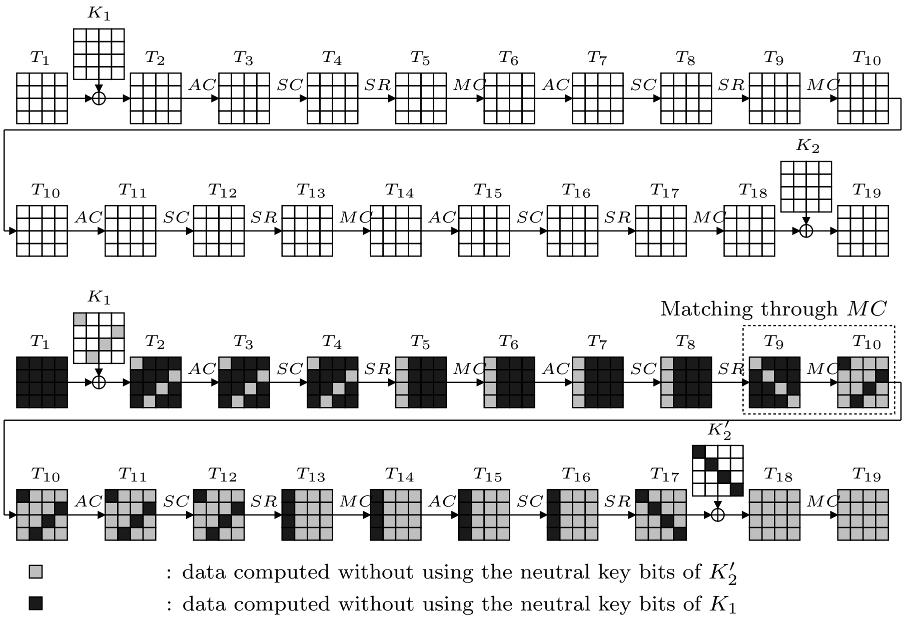


Figure 4.6: 4-round partial matching of LED-128

- **Evaluation.** The complexity of the attack C_{comp} is estimated as

$$C_{comp} = \underbrace{2^{96}(2^{16} + 2^{16})}_{\text{MITM}} + \underbrace{2^{96}}_{\text{Key testing}} \approx 2^{112}.$$

The required memory is 2^{19} byte ($\approx 2^{16} \cdot 6$ bytes). For the backward chunk, only 16 bits of the plaintext are affected by $K_1[0, 7, 10, 13]$. Thus, when the start state is properly chosen similar to the attack on XTEA, the number of required plaintext/ciphertext pairs is estimated as 2^{16}.

MITM Attack on 8-round LED-64.

We consider the 8-round variant of LED-64 starting from the round 1. Let K_a and K_b be the 32-bit left and right two columns of K, respectively, i.e., $K_a = K[0, 1, 4, 5, 8, 9, 12, 13]$ and $K_b = K[2, 3, 6, 7, 10, 11, 14, 15]$. Then the 64-bit key XOR of K is divided into two 32-bit key XORs of K_a and K_b. FF denotes the functions from rounds 1 to 4 (i.e., $F[1, 4]$ of LED-64) including key addition of K_b, and FB denotes the first and last key addition of K_a. By using K_a and K_b, two chunks FB and FF can be computed independently (see Fig. 4.5).

Suppose that $\mathcal{K}_{(1)}$ is $K[10, 15] \in K_b$ and $\mathcal{K}_{(2)}$ is $K'[0, 13] \in K_a$, where $K' = MC^{-1}(K)$. Then, the 4-round partial matching can be constructed similar to the partial matching for the attack on the reduced LED-128. The complexity of the attack C_{comp} is estimated as

$$C_{comp} = \underbrace{2^{48}(2^{8} + 2^{8})}_{\text{MITM}} + \underbrace{2^{32}}_{\text{Key testing}} \approx 2^{56}.$$

The required memory is 2^{11} bytes and the required data is 2^{8} chosen plaintext/ciphertext pairs.

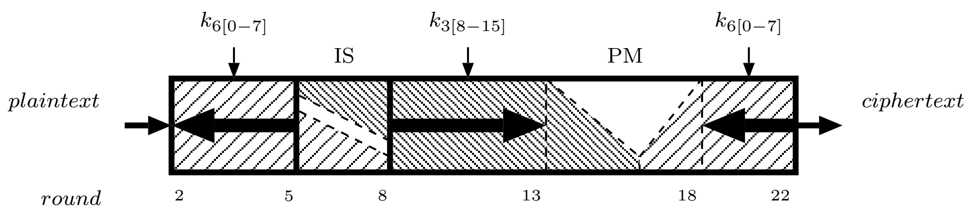


Figure 4.7: Overview of the 21-round attack on Piccolo-128

4.4.3 Security of Piccolo against MITM Attack

We show MITM attacks on the 21-round and the 14-round reduced Piccolo-128 and Piccolo-80, respectively.

MITM Attack on 21-Round Piccolo-128.

For the 21-round variant of Piccolo-128 starting from the round 2, using neutral key bits $\mathcal{K}_{(1)}(= k_{3[8-15]})$ and $\mathcal{K}_{(2)}(= k_{6[0-7]})$, two chunks $F[9,13]$ and $F^{-1}[19,22] \circ F^{-1}[2,5]$ can be computed independently (see Fig. 4.7). We give detailed explanations for the partial matching and the initial structure used in this attack, and the estimated attack complexity as follows.

- **Partial Matching.** For $F[14,18]$, in the forward process, X_1^{17} is obtained from S^{14} without using the value of $k_{6[0-7]}$. On the other hand, X_1^{17} is also computed from S^{19} without using the value of $k_{3[8-15]}$ in the backward process. Thus, the 5-round partial matching works (see Fig. 4.8).

- **Initial Structure.** For $F[6,8]$, we aim to exchange the positions of $k_{6[0-7]}$ for $k_{3[8-15]}$. Here, differences of $k_{6[0-7]}$ in the forward process and $k_{3[8-15]}$ in the backward process are independent in $F[6,8]$. It means that one differential trail does not affect the others, since these trails do not share any non-linear elements. Thus, these values are exchangeable by splitting $F[6,8]$ as illustrated in Fig. 4.9. This can be seen as the 3-round bicliques, and finding independent differentials directly leads the construction of the initial structure as mentioned in [70].

- **Evaluation.** The complexity of the attack on the 21-round reduced Piccolo-128 C_{comp} is estimated as

$$C_{comp} = \underbrace{2^{112}(2^8 + 2^8)}_{\text{MITM}} + \underbrace{2^{112}}_{\text{Key testing}} \approx 2^{121}.$$

The required memory is 2^9 bytes ($= 2^8 \cdot 2$ bytes), which is the cost of the table used in the MITM stage. The number of required plaintext/ciphertext pairs depends on the range of $F^{-1}[2,8]$, because $k_{6[0-7]}$ is included in the initial structure. Due to the 4-round full diffusion property, the required data is 2^{64} plaintext/ciphertext pairs, which is called a code book attack. In case the code book attack is not allowed, the number of attacked rounds might be decreased.

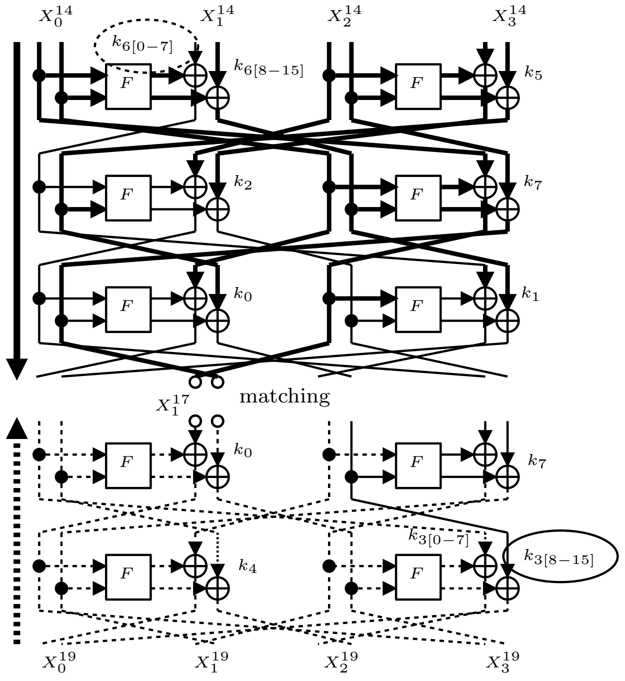


Figure 4.8: 5-round PM of Piccolo-128

MITM Attack on 14-Round Piccolo-80.

For the MITM attack on the 14-round variant of Piccolo-80 starting from the round 5, using neutral key bits $\mathcal{K}_{(1)}(= k_{0[0-7]})$ and $\mathcal{K}_{(2)}(= k_{4[8-15]})$, two chunks $F[5,8] \circ F[15,18]$ and $F^{-1}[13,14]$ can be computed independently. For $F[9,12]$, the 4-round partial matching is constructed as illustrated in Fig. 4.10. The complexity of the attack on the 14-round reduced Piccolo-80 C_{comp} is estimated as

$$C_{comp} = \underbrace{2^{64}(2^8 + 2^8)}_{\text{MITM}} + \underbrace{2^{72}}_{\text{Key testing}} \approx 2^{73}.$$

The required memory is 2^8 bytes $(= 2^8 \cdot 1$ bytes), which is the cost of the table used in the MITM stage. The number of required plaintext/ciphertext pairs depends on the range of $F[15,18]$, i.e., size of the possible output values of $F[15,18]$ during the MITM stage. In general, if neutral key bits $k_{0[0-7]}$ does not affect the all bits of the output of $F[15,18]$, it is possible to avoid code book attack by fixing some bits of the output similar to the AES attack [70]. However, $k_{0[0-7]}$ affects all bits of the output due to the 4-round full diffusion property. Thus, the required data is 2^{64} plaintext/ciphertext pairs. In our search, such 14-round attack without relying on the code book cannot be found. Thus, we expect that the number of attacked rounds may be slightly reduced in the non-code book attack on the reduced Piccolo-80.

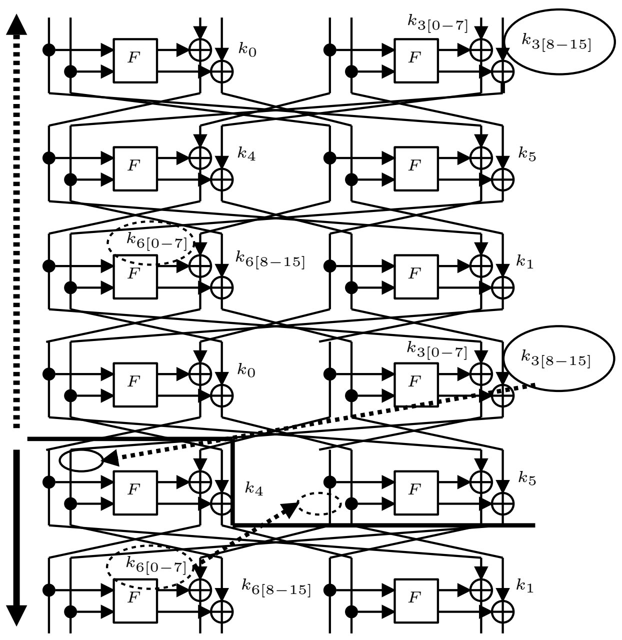


Figure 4.9: 3-round IS of Piccolo-128

4.5 Discussion

This section discusses the security of lightweight block ciphers against the speed-up keysearch based on the MITM attack. Then we compare our results with the previous attack results on our target ciphers.

4.5.1 Security against Speed-up Keysearch Based on MITM Attack

The speed-up keysearch based on the MITM attack is the recently proposed novel technique, which is known as the first attack for the full AES [70]. It makes use of *matching with precomputation* in conjunction with a *biclique* which is a formal concept of the initial structure. In general, it works slightly faster than the exhaustive key search, i.e., by a factor of 2 to 5, since it essentially tests all possible keys. This cryptanalysis is naturally applicable to lightweight block ciphers. Indeed, it has been applied to HIGHT [78], whose attacks is slightly faster than that of the exhaustive key search. As mentioned in [79], for many block ciphers, one also can speed up the exhaustive keysearch with simple techniques, e.g., *distributive technique* and *early abort technique.* Therefore, it is debatable whether such a marginal improvement regarding the time complexity should be considered as a real attack in terms of exploiting algorithmic weaknesses. However, it is still meaningful from the view of the security evaluation of such ciphers. In the following, we roughly estimate the required time complexity for the speed-up keysearch of the lightweight block ciphers, XTEA, LED and Piccolo, assuming that the code book is allowed to use and the rounds of the partial matching and the initial structure can be omitted, i.e., the cost of the recomputation is regarded as zero in these rounds. Due to such strong assumptions, our estimations may not be accurate but provide indications of

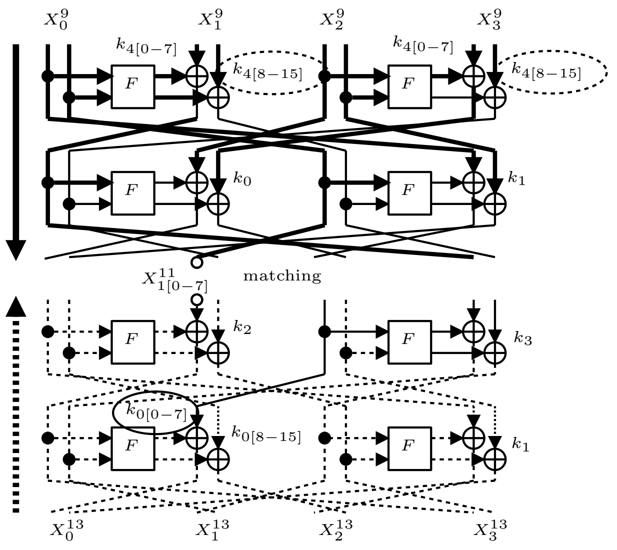


Figure 4.10: 4-round partial matching of Piccolo-80

the security evaluations of the speed-up keysearch based on the MITM attack.

For XTEA, assuming the 12-round partial matching and two 6-round independent chunks, the time complexity for the speed-up keysearch of the full XTEA is estimated as $2^{127.32}(= 2^{128} \times ((64 - 12 - 12)/64))$. For LED-64/128, the 4-round partial matching is constructed. Exploiting the two 4-round independent chunks, the time complexities for the speed-up keysearch of the full LED-64/128 are roughly estimated as $2^{127.59} = (2^{128} \times ((48-4-8)/48))$ and $2^{79.32}(= 2^{80} \times ((32-4-8)/32))$, respectively. For Piccolo-80/128, the 3-round initial structure and the 5-round partial matching are constructed as described in Section 4.4.3. Assuming the two 5-round independent chunks, the time complexity for the speed-up keysearch of the full Piccolo-128 is roughly estimated as $2^{126.74}(= 2^{128} \times ((31 - 3 - 5 - 10)/31))$. In a similar way, the time complexity of the full Piccolo-80 is roughly estimated as $2^{78.53}(= 2^{80} \times ((25 - 3 - 5 - 8)/25))$.

We emphasize that the above observations are rough estimations of the speed-up keysearch based on the MITM attack. Thus, the time complexities seem to be improved by analyzing deeply [3]. However, since these are marginal improvements in time complexity compared to the exhaustive key search, we do not claim them to be real attacks based on algorithmic weaknesses.

4.5.2 Comparison with Other Cryptanalysis Results

We compare our MITM attacks to the previously proposed attacks for each cipher. Note that we focus only on the single-key setting, which is the practical setting of fields in where lightweight cipher are needed, e.g., RFID tags and sensor nodes.

For XTEA, the previously best attacks regarding the number of the attacked rounds are the impossible differential attack and the MITM attack on the 23-round

[3]This types of the speed-up keysearch on the reduced Piccolo-80/128 has been estimated in [80].

variants [74, 75]. Thus, our MITM attack, which is the 29-round attack, greatly improves their attacks with respect to the number of attacked rounds.

For LED-64/128 and Piccolo-80/128, there exist only designers' self evaluations, i.e., no external cryptanalysis has been published so far. According to [20], the maximum differential probability and the best linear hull probability of 4 rounds of LED-64/128 are both upper bounded by 2^{-32}. Thus, it was implicitly claimed that there is no useful differentials or linear hulls over 8 rounds of LED-64/128 in the single-key setting, though the actual attack has not been presented so far. By using the equivalent transformation technique, we showed the MITM attacks on the 8-round LED-64 and the 16-round LED-128. While the designers showed chosen key distinguishers for the 15-round LED-64 and the 27-round LED-128, our MITM attacks can be regarded as the best attacks on the reduced LED in the single-key setting.

For Piccolo-80/128, according to [21], the 9-round Piccolo-80/128 have a sufficient security level against differential and the linear cryptanalysis by the evaluations based on the number of the active F-functions. On the other hand, our MITM attacks work on the 14-round Piccolo-80 and the 21-round Piccolo-128, though both attacks require full plaintext/ciphertext pairs, which are called code book attacks. These results imply that MITM-type attacks are more effective than differential and linear cryptanalysis for both LED-64/128 and Piccolo-80/128 in the single-key setting, because the number of rounds to be attacked by the MITM attack is much larger than those of differential and linear cryptanalysis.

Therefore it is demonstrated that MITM attacks are the most powerful attacks on these lightweight ciphers in the single-key setting.

4.6 Conclusion

We have analyzed the security of several lightweight block ciphers that have the direct-key expansion against the meet-in-the-middle attack. While a simple key expanding function like the direct-key expansion leads compact implementation, its security analysis has not been studied well so far. On the other hand, since MITM attack mainly exploits low key-dependency that large parts of the cipher are independent from some key bits, a block cipher having the direct-key expansion is likely to be vulnerable to MITM attack. Moreover, since the MITM attack does not require related-keys, i.e., it works in the single-key setting, its security analysis is considered to be more important than related-key attacks.

In this chapter, we have shown new MITM attacks on several lightweight block ciphers that have the direct-key expansion. Combined with new techniques such as equivalent transformation technique, we have presented the MITM attacks on 29 (out of 64), 8 (out of 32), 16 (out of 48), 14 (out of 25) and 21 (out of 31) rounds of XTEA, LED-64, LED-128, Piccolo-80 and Piccolo-128, respectively. Note that all of our attacks presented in this chapter are the best or the first attacks in literature. For instance, the attack on the 29-round reduced XTEA is the best attack with respect to the number of attacked rounds. These results imply that MITM attack is actually effective for lightweight block ciphers. Thus, its security analysis is crucial, even if the cipher is expected to be secure against the other attacks.

Chapter 5

All Subket Recovery Attack on Block Ciphers : Extending Meet-in-the Middle Approach

We revisit meet-in-the-middle (MITM) attacks on block ciphers. Despite recent significant improvements of the MITM attack, its application is still restrictive. In other words, most of the recent MITM attacks work only on block ciphers consisting of a bit permutation based key schedule such as KTANTAN, GOST, IDEA, XTEA, LED and Piccolo. In this chapter, we extend the MITM attack so that it can be applied to a wider class of block ciphers. In our approach, MITM attacks on block ciphers consisting of a complex key schedule can be constructed. We regard all subkeys as independent variables, then transform the game that finds the user-provided key to the game that finds all independent subkeys. We apply our approach called all subkeys recovery (ASR) attack to block ciphers employing a complex key schedule such as CAST-128, SHACAL-2, KATAN, FOX128 and Blowfish, and present the best attacks on them with respect to the number of attacked rounds in literature. Moreover, since our attack is simple and generic, it is applied to the block ciphers consisting of any key schedule functions even if the key schedule is an ideal function.

5.1 Introduction

Meet-in-the-middle (MITM) attack, originally introduced in [56], is a generic cryptanalytic technique for block ciphers. It was extended to preimage attacks on hash functions, and several novel techniques to extend the attack have been developed [16, 81, 82, 83, 84, 85]. Then, it has been shown that those advanced techniques are also applied to block ciphers [9, 86, 70, 71].

Since the MITM attack mainly exploits a low key-dependency in a key schedule, it works well for a block cipher having a simple key schedule such as a key schedule based only on a bit permutation. In fact, most of the recent MITM attacks were applied to ciphers having a simple key schedule such as KTANTAN, GOST, IDEA, XTEA, LED and Piccolo [9, 86, 71, 87, 88]. However, as far as we know, no results have been known so far for block ciphers consisting of a complex key schedule[1] except for the recent attack on AES [70]. In general, the MITM attack at least requires two sets of neutral key bits, which are parts of secret key bits, to compute two functions independently. For a simple key schedule such as a permutation based key schedule, since each subkey is directly derived from some secret key bits, it is relatively simple to find "good" neutral key bits. Yet, for a cipher equipped with a complex key schedule, finding neutral key bits, which is the core technique of the MITM attack, is likely to be complicated and specific. For instance, the MITM attack on the 8-round reduced AES-128 in [70] looks complicated and specific, i.e., it seems difficult to directly apply their technique to other block ciphers not having the AES key schedule.

In this chapter, we extend the MITM attack, then present a generic and simple approach to evaluate the security of ciphers employing a complex key schedule against the MITM attack. The basic concept of our approach is converting the game of finding the user-provided key (or the secret key) to the game of finding all subkeys so that the analysis can be independent from the structure of the key schedule, by regarding all subkeys as independent variables. This simple conversion enables us to apply the MITM attack to a cipher using any key schedule without analyzing the key schedule. We refer this attack as all subkeys recovery (ASR) attack for simplicity, while our approach is a variant of the MITM attack. We first apply the ASR attack to CAST-128, Blowfish and FOX128 in a straightforward way, then show the best attacks on them with respect to the number of attacked rounds in literature. Moreover, to construct more efficient attack, we present how to efficiently find useful state that contains a smaller number of subkey bits by analyzing internal components, then apply it to SHACAL-2 and KATAN family. The attacks presented in this chapter are summarized in Table 10.1 (see also Table 5.2). We emphasize that our attack can be applied to any block cipher having any key schedule function even if the key schedule is an ideally random function. This implies that our approach gives generic lower bounds on the security of several block ciphers against the MITM attacks.

5.2 Notation

The following notation will be used throughout this chapter:

$a \| b$	:	Concatenation.
$\lvert a \rvert$	:	Bit size of a.
$\ggg$ or $\lll$	:	Right or left bit rotation in 32-bit word.
$\oplus$	:	Bitwise logical exclusive OR (XOR) operation.
$\boxplus$	:	Addition modulo 2^{32} operation.
$\boxminus$	:	Subtraction modulo 2^{32} operation.

5.3 All Subkeys Recovery (ASR) Attack

In this section, the basic concept of the all subkeys recovery (ASR) attack is introduced. First, we briefly present the basic concept, then, the detailed procedure of

[1]In this chapter, we refer a complex key schedule as a non-permutation based key schedule such as a key schedule having non-linear components.

the basic ASR attack is given. Finally, we show an ASR attack on the balanced Feistel network as a concrete example.

5.3.1 Basic Concept

The previous MITM attack aims to determine the user-provided key by finding good neutral key bits which are parts of the user-provided key. In order to find good neutral key bits, an attacker needs to thoroughly analyze the key schedule. In general, this makes the analysis complex and specific, and it is difficult to evaluate the security of a wide class of block ciphers, including ciphers having a complex key schedule, against the MITM attack.

The basic concept of the ASR attack is to convert the game of finding the user-provided key to the game of finding all subkeys, regarding all subkeys as independent variables. Note that an attacker is able to encrypt/decrypt any plaintexts/ciphertexts by using all subkeys, even if the user-provided key is unknown. In addition, if a key schedule is invertible, then the user-provided key is obtained from all subkeys.

In the ASR attack, analyzing the key schedule is not mandatory, since we only treat subkeys (not secret key). Obviously, this makes analysis simpler than finding good neutral key bits. In fact, the ASR attack depends only on the sizes of the secret key and the round keys, and the structure of the data processing part such as balanced Feistel network and SPN (substitution-permutation-network).

Moreover, in the ASR attack, the underlying key schedule can be treated as an ideal function. In other words, our attack works even if the key schedule is an ideal function. A concrete block cipher employs an weaker key schedule than an ideal one. Thus, the number of attacked rounds may be extended by thoroughly analyzing the key schedule. However, even without analyzing the key schedule, we show several best attacks on several block ciphers in the single-key setting in the following sections. Thanks to the MITM approach, our attack requires extremely low data requirement, though it requires a lot of memory which is the same order as the time complexity. We may employ memoryless collision search [89] to reduce the memory requirements, while it requires a slightly larger computations, i.e., the time complexity is multiplied by a small constant.

5.3.2 Recovering All Subkeys by MITM Approach (Basic Attack)

Let us explain the basic procedure of the ASR attack. As explained in the previous section, we regard all subkeys in the cipher as independent variables. Then, we apply the MITM approach to determine all subkeys that map a plaintext to the ciphertext encrypted by the (unknown) secret key. Suppose that n-bit block cipher E accepting a k-bit secret key K consists of R rounds, and an ℓ-bit subkey is introduced each round (see Fig. 5.1).

First, an attacker determines an s-bit matching state S. The state S can be computed from a plaintext P and a set of subkey bits $\mathcal{K}_{(1)}$ by a function $\mathcal{F}_{(1)}$ as $S = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$. Similarly, S can be computed from the ciphertext C and another set of subkey bits $\mathcal{K}_{(2)}$ by a function $\mathcal{F}_{(2)}$ as $S = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$. $\mathcal{K}_{(3)}$ denotes a set of the remaining subkey bits, i.e., $|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}| + |\mathcal{K}_{(3)}| = R \cdot \ell$. By using those $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$, the attacker can independently compute $\mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$ and $\mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$. Note

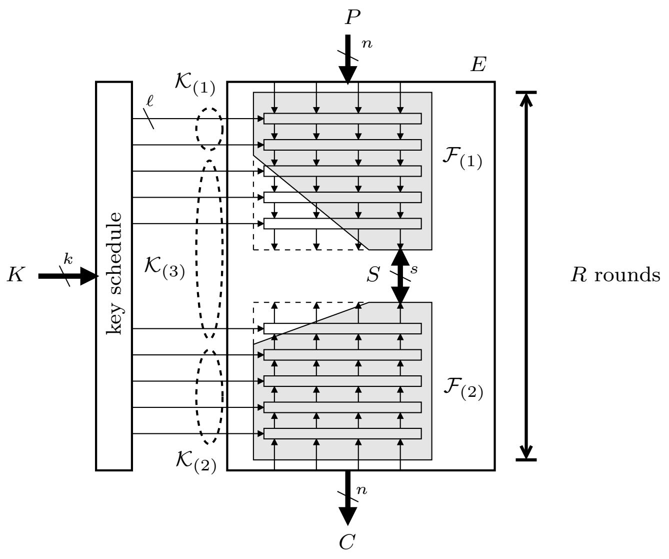


Figure 5.1: Basic concept of ASR attack

that the equation $\mathcal{F}_{(1)}(P, \mathcal{K}_{(1)}) = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$ holds when the guessed subkey bits $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are correct. Due to parallel guesses of $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$, we can efficiently check if the guessed key bits are correct. After this process, it is expected that there will be $2^{R \cdot \ell - s}$ key candidates. Note that the number of key candidates can be reduced by parallel performing the matching with additional plaintext/ciphertext pairs. In fact, using N plaintext/ciphertext pairs, the number of key candidates is reduced to $2^{R \cdot \ell - N \cdot s}$, as long as $N \leq (|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}|)/s$. Finally, the attacker exhaustively searches the correct key from the surviving key candidates. The required computations (i.e. the number of encryption function calls) of the attack in total C_{comp} is estimated as

$$C_{comp} = \max(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N + 2^{R \cdot \ell - N \cdot s}. \tag{5.1}$$

The number of required plaintext/ciphertext pairs is $\max(N, \lceil (R \cdot \ell - N \cdot s)/n \rceil)$. The required memory is about $\min(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N$ blocks, which is the cost of the table used for the matching. Obviously, the ASR attack works faster than the brute force attack when Eq.(5.1) is less than 2^k, which is required computations for the brute force attack. For simplicity, we omit the cost of memory access for finding a match between two lists, assuming that the time complexity of one table look-up is negligible compared to that of one computation of $\mathcal{F}_{(1)}$ or $\mathcal{F}_{(2)}$. The assumption is quite natural in most cases. However, strictly speaking, those costs should be considered. In other words, the cost of $(\max(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N)$ memory accesses is added to Eq.(5.1).

If the number of attacked rounds R and the size of the matching state s are fixed, the time complexity and the memory requirement are dominated by $|\mathcal{K}_{(1)}|$ and $|\mathcal{K}_{(2)}|$. Thus, smaller $|\mathcal{K}_{(1)}|$ and $|\mathcal{K}_{(2)}|$ lead to more efficient attacks with respect to the time and memory complexity. Therefore, the key of the ASR attack is to find the matching state S computed by the smallest $\max(|\mathcal{K}_{(1)}|, |\mathcal{K}_{(2)}|)$.

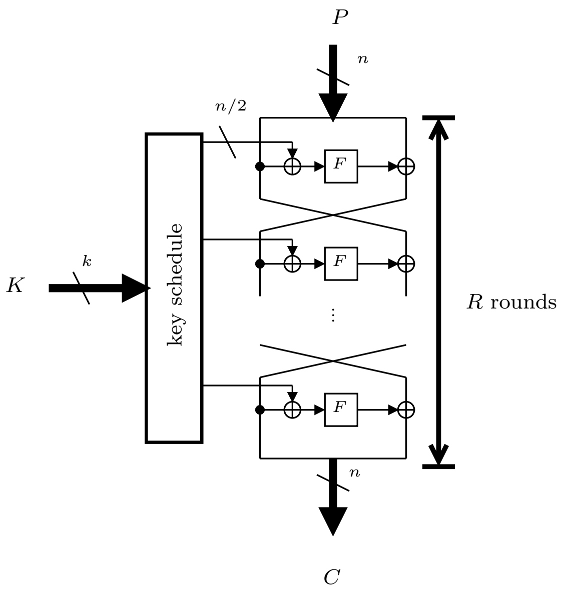


Figure 5.2: Balanced Feistel network

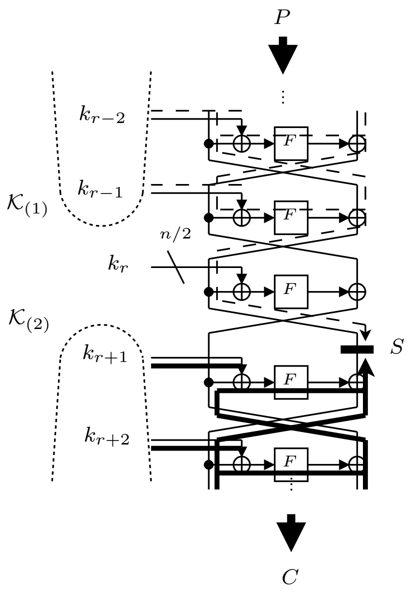


Figure 5.3: Partial matching

5.3.3 ASR Attack on Balanced Feistel Networks

Let us show examples of the ASR attack. Suppose that an example cipher E with n-bit block and k-bit secret key consists of R rounds of the balanced Feistel network as illustrated in Fig. 5.2. Let the round function F be an $(n/2)$-bit keyed bijective function. Here, for simplicity, we assume that an $(n/2)$-bit subkey is introduced before F each round. Also, k_i denotes the i-th round subkey.

As depicted in Fig. 5.3, an $(n/2)$-bit state S can be computed independently from $(n/2)$-bit subkey k_r. In other words, S can be computed from P and $\mathcal{K}_{(1)} \in \{k_1, k_2, ..., k_{r-1}\}$. Also, S can be obtained from C and $\mathcal{K}_{(2)} \in \{k_{r+1}, k_{r+2}, ..., k_R\}$.

When $n = k/2$ (e.g., a 128-bit block cipher accepting a 256-bit key), 7 rounds of E can be attacked in a straightforward manner. In this attack, both $\mathcal{F}_{(1)}$ and $\mathcal{F}_{(2)}$ are composed of 3 rounds of E, and thus the sizes of $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are both $3n/2$ bits. As explained in Section 5.3.2, using six plaintext/ciphertext pairs, the total time complexity C_{comp} is estimated as

$$\begin{aligned} C_{comp} &= \max(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N + 2^{R\cdot\ell - N\cdot s} \\ &= 2^{3n/2} \times 6 + 2^{7\cdot n/2 - 6\cdot n/2} \approx 2^{3n/2} \quad (= 2^{3k/4} \ll 2^k) \end{aligned}$$

The required memory is around $6 \times 2^{3n/2}$ blocks. Since C_{comp} is less than $2^{2n} (= 2^k)$, the attack works faster than the exhaustive key search. Note that the number of attacked rounds might be extended by exploiting the subkey relations. Thus, the number of attacked rounds 7 is considered as the lower bounds on the security of this modelled cipher against the ASR attack.

Similarly to this, when $n = k$ (e.g., a 128-bit block cipher accepting a 128-bit key), the attack on at least 3 rounds of E is constructed. In this case, $\mathcal{F}_{(1)}$ and $\mathcal{F}_{(2)}$ consist of 1 round of E, and the sizes of $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are both $n/2$ bits. Therefore, using 3 plaintext/ciphertext pairs (i.e. $N = 3$), the required time complexity C_{comp}

Table 5.1: Basic parameters of our target ciphers

algorithm	block size (n)	key size (k)	subkey size (ℓ)	# rounds (R)
CAST-128 [22]	64	$40 \leq k \leq 128$	37	12 ($k \leq 80$), 16 ($k > 80$)
Blowfish [25]	64	$128 \leq k \leq 448$	32	16
Blowfish-8R* [90]	64	$128 \leq k \leq 192$	32	8
SHACAL-2 [23]	256	$k \leq 512$	32	64
KATAN32/48/64 [24]	32/48/64	80	2	254
FOX128 [26]	128	256	128	16

$*$ Fewer iteration version of Blowfish specified in [25] as a possible simplification.

is estimated as

$$C_{comp} = \max(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(1)}|}) \times N + 2^{R\cdot\ell - N\cdot s} = 2^{n/2} \times 3 + 1 \quad (= 2^{k/2} \times 3 + 1 \ll 2^k).$$

The required memory is around $3 \times 2^{n/2}$ blocks. Consequently, the ASR attack works faster than the brute force attack, which requires about 2^n computations.

Roughly speaking, when Eq.(5.1) is less than 2^k, the ASR attack works faster than the brute force attack. Therefore, the necessary condition for the basic ASR attack is that the size of subkey is less than the size of secret key.

5.4 Basic ASR Attacks on CAST-128 and Blowfish

The generic attack on a balanced Feistel network explained in the previous section can be directly applied to a concrete block cipher. In this section, we apply the basic ASR attacks to CAST-128 and Blowfish. In those attacks, the round function F is assumed to be any function even ideal. However, in the case of a concrete cipher, the underlying round function F is specified, i.e., it can be analyzed. By deeply analyzing the round function, the number of attacked rounds may be increased. Such advanced techniques are introduced in the next sections.

The basic parameters of the ciphers analyzed in this chapter are listed in Table 5.1. Table 5.2 shows the parameters of our (ASR) attacks in this chapter.

5.4.1 Descriptions of CAST-128 and Blowfish

Description of CAST-128.

CAST-128 [22] is a 64-bit Feistel block cipher accepting a variable key size from 40 up to 128 bits (but only in 8-bit increments). The number of rounds is 16 when the key size is longer than 80 bits. First, the algorithm divides the 64-bit plaintext into two 32-bit words L_0 and R_0, then the i-th round function outputs two 32-bit data L_i and R_i as follows:

$$L_i = R_{i-1}, R_i = L_{i-1} \oplus F_i(R_i, K_i^{rnd}),$$

where F_i denotes the i-th round function and K_i^{rnd} is the i-th round key consisting of a 32-bit masking key K_{m_i} and a 5-bit rotation key K_{r_i}. Each round function

Table 5.2: Parameters of our attacks presented in this chapter

algorithm	# attacked rounds	$\|\mathcal{K}_{(1)}\|$ (forward)	$\|\mathcal{K}_{(2)}\|$ (backward)	$\|\mathcal{K}_{(3)}\|$ (remains)	s	attacked key size
CAST-128	7	111	111	37	32	$\mathbf{120} \leq k \leq 128$
Blowfish[†]	16	256	288	32	32	$\mathbf{292} \leq k \leq 448$
Blowfish-8R[†]	8	128	160	32	32	$\mathbf{163} \leq k \leq 192$
SHACAL-2	41	484	492	336	4	$\mathbf{485} \leq k \leq 512$
KATAN32	110	68	70	82	1	80
KATAN48	100	71	71	58	1	80
KATAN64	94	71	71	46	1	80
FOX128	5	224	224	192	32	256

† Known F-function setting.

F_i consists of four 8 to 32-bit S-boxes, a key dependent rotation, and logical and arithmetic operations (addition, subtraction and XOR). F_i has three variations, and the positions of three logical or arithmetic operations are varied in each round. However, we omit the details of F_i, since, in our analysis, it is regarded as the random function that outputs a 32-bit random value from a 32-bit input R_i and a 37-bit key K_i^{rnd}.

Description of Blowfish.

Blowfish [25] is a 16-round Feistel block cipher with 64-bit block and variable key size from 128 up to 448 bit. Several possible simplifications of Blowfish were also described in [25]. We refer one of them that is an 8-round variant (fewer iterations) of Blowfish accepting less than 192 bits of key as Blowfish-8R. First, Blowfish divides a 64-bit plaintext into two 32-bit state L_0 and R_0. Then the i-th round output state $(L_i||R_i)$ is computed as follows:

$$R_i = L_{i-1} \oplus K_i^{rnd}, \qquad L_i = R_{i-1} \oplus F(L_i \oplus K_i^{rnd}),$$

where K_i^{rnd} is a 32-bit round key at round i. The F-function F consists of four 8×32 key dependent S-boxes. Note that, in the last round, the 64-bit round key is additionally XORed with the 64-bit state. In this chapter, we assume that F-function is known by an attacker, and it is a random 32-bit function. The assumption, i.e., the known F-function setting, has already been used in [91, 92].

5.4.2 ASR Attack on 7-Round Reduced CAST-128

The generic attack on a balanced Feistel network can be directly applied to a variant of CAST-128 which accepts a more than 114 bits key. This variant consists of 16 rounds, since the key size is longer than 80 bits. While the size of each subkey of the CAST-128 is 37 bits including a 32-bit masking key and a 5-bit rotation key, the above explained attack still works faster than the exhaustive key search. For the 7-round reduced CAST-128, we use $|\mathcal{K}_{(1)}| = |\mathcal{K}_{(2)}| = 111(= 37 \cdot 3)$, since each of $\mathcal{F}_{(1)}$ and $\mathcal{F}_{(2)}$ consists of three rounds of the CAST-128. Consequently, using six plaintext/ciphertext pairs, the total time complexity C_{comp} for the attack on the

7-round reduced CAST-128 is estimated as follows:

$$\begin{aligned} C_{comp} &= \max(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N + 2^{R\cdot\ell - N\cdot s} \\ &= 2^{111} \times 6 + 2^{7\cdot 37 - 6\cdot 32} = 2^{111} \times 6 + 2^{67} \approx 2^{114}. \end{aligned}$$

The number of required known plaintext/ciphertext pairs is only 6 (= max(6, $\lceil(258 - 6\cdot 32)/64\rceil$), and the required memory is about 2^{114} (= $\min(2^{111}, 2^{111}) \times 6$) blocks. Recall that our attack works faster than the exhaustive key search only when the key size of the reduced CAST-128 is more than 114 bits. As far as we know, the previous best attack on the reduced CAST-128 was for only 6 rounds in the single-key setting [93][2]. Thus, surprisingly, this simple attack exceeds the previously best attack on the reduced CAST-128 with respect to the number of attacked rounds.

5.4.3 ASR Attack on Full Blowfish in Known F-function Setting

In this section, we apply the ASR attack on Blowfish block cipher in the known F-function setting as with [91, 92].

For the full (16-round) Blowfish, we choose R_8 as the 32-bit matching state, i.e., $s = 32$. Then $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ include 256 (= 32×8) and 288 (= 32×9) bits of subkeys, respectively, and $\mathcal{K}_{(3)} = 32$. When $N = 9$ ($\leq (256 + 288)/32$), the time complexity for finding all subkeys is estimated as

$$C_{comp} = \max(2^{256}, 2^{288}) \times 9 + 2^{576 - 9\cdot 32} = 2^{292}.$$

The number of required data is only 9 (=max(9, $\lceil(576 - 9\cdot 32)/64\rceil$)) known plaintext/ciphertext pairs, and required memory is about 2^{260} (=$\min(2^{256}, 2^{288}) \times 9$) blocks. In this setting, the attack works faster than the exhaustive key search when the key size is more than 292 bits.

Similarly to the attack on the full Blowfish, for the full (8-round) Blowfish-8R, we choose R_4 as the 32-bit matching state, i.e., $s = 32$. Then $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ include 128 (= 32×4) and 160 (= 32×5) bits of subkeys, respectively, and $\mathcal{K}_{(3)} = 32$. When $N = 5$ ($\leq (128 + 160)/32$), the time complexity for finding all subkeys is estimated as

$$C_{comp} = \max(2^{128}, 2^{160}) \times 5 + 2^{320 - 5\cdot 32} = 2^{163}.$$

The number of required data is only 5 (=max(5, $\lceil(320 - 5\cdot 32)/64\rceil$)) known plaintext/ciphertext pairs, and the required memory is about 2^{131} (=$\min(2^{128}, 2^{160}) \times 5$) blocks. In this setting, the attack works faster than the exhaustive key search when the key size is more than 163 bits.

Note that these attacks are the first results on the full Blowfish with all key classes in the known F-function setting, while the attacks presented in [91, 92] work only in the weak key setting, i.e., weak F-functions.

5.5 Application to SHACAL-2

In this section, we apply the ASR attack to SHACAL-2 block cipher. After a brief description of SHACAL-2, we present the basic ASR attack on the reduced

[2]The differential attacks on the 8- and 9-round reduced CAST-128 in weak-key setting were presented in [94].

SHACAL-2 without analyzing the internal functions of the cipher. Then, by analyzing the functions of the cipher, we show the matching state that contains fewer bits of subkeys. Finally, we demonstrate an advanced ASR attack by using this matching state. Recall that the basic ASR attack regards internal components such as an F-function of the balanced Feistel network as a black-box function, while the advanced ASR attack analyzes the internal components.

5.5.1 Description of SHACAL-2

SHACAL-2 [23] is a 256-bit block cipher based on the compression function of SHA-256 [39]. It was submitted to the NESSIE project and selected to be in the NESSIE portfolio [95].

SHACAL-2 inputs the plaintext to the compression function as the chaining variable, and inputs the key to the compression function as the message block. First, a 256-bit plaintext is divided into eight 32-bit words A_0, B_0, C_0, D_0, E_0, F_0, G_0 and H_0. Then, the state update function updates eight 32-bit variables, A_i, B_i, ..., G_i, H_i in 64 steps as follows:

$$\begin{aligned} T_1 &= H_i \boxplus \Sigma_1(E_i) \boxplus Ch(E_i, F_i, G_i) \boxplus K_i \boxplus W_i, \\ T_2 &= \Sigma_0(A_i) \boxplus Maj(A_i, B_i, C_i), \\ A_{i+1} &= T_1 \boxplus T_2,\ B_{i+1} = A_i,\ C_{i+1} = B_i,\ D_{i+1} = C_i, \\ E_{i+1} &= D_i \boxplus T_1,\ F_{i+1} = E_i,\ G_{i+1} = F_i,\ H_{i+1} = G_i, \end{aligned}$$

where K_i is the i-th step constant, W_i is the i-th step key (32-bit), and the functions Ch, Maj, Σ_0 and Σ_1 are given as follows:

$$\begin{aligned} Ch(X, Y, Z) &= XY \oplus \overline{X}Z, \\ Maj(X, Y, Z) &= XY \oplus YZ \oplus XZ, \\ \Sigma_0(X) &= (X \ggg 2) \oplus (X \ggg 13) \oplus (X \ggg 22), \\ \Sigma_1(X) &= (X \ggg 6) \oplus (X \ggg 11) \oplus (X \ggg 25). \end{aligned}$$

After 64 steps, the function outputs eight 32-bit words A_{64}, B_{64}, C_{64}, D_{64}, E_{64}, F_{64}, G_{64} and H_{64} as the 256-bit ciphertext. Hereafter p_i denotes the i-th step state, i.e., $p_i = A_i||B_i||...||H_i$.

The key schedule of SHACAL-2 takes a variable length key up to 512 bits as the inputs, then outputs 64 32-bit step keys. First, the 512-bit input key is copied to 16 32-bit words W_0, W_1, ..., W_{15}. If the size of the input key is shorter than 512 bits, the key is padded with zeros. Then, the key schedule generates 48 32-bit step keys $(W_{16}, ..., W_{63})$ from the 512-bit key $(W_0, ..., W_{15})$ as follows:

$$W_i = \sigma_1(W_{i-2}) \boxplus W_{i-7} \boxplus \sigma_0(W_{i-15}) \boxplus W_{i-16}, (16 \leq i < 64),$$

where the functions $\sigma_0(X)$ and $\sigma_1(X)$ are defined by

$$\begin{aligned} \sigma_0(X) &= (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3), \\ \sigma_1(X) &= (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10). \end{aligned}$$

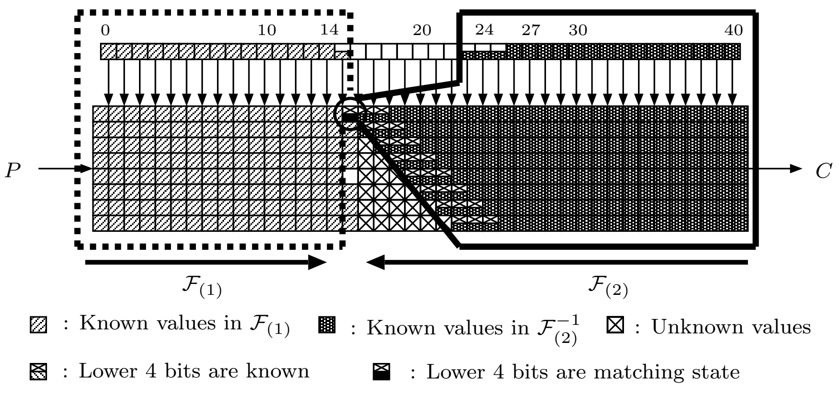

0
10
14
20
24
27
30
40
P
C
$\mathcal{F}_{(1)}$
$\mathcal{F}_{(2)}$
▨ : Known values in $\mathcal{F}_{(1)}$
▩ : Known values in $\mathcal{F}^{-1}_{(2)}$
⊠ : Unknown values
▧ : Lower 4 bits are known
◙ : Lower 4 bits are matching state

Figure 5.4: Overview of 41 round attack on reduced SHACAL-2

5.5.2 Basic ASR Attack on 37-Step Reduced SHACAL-2

We directly apply the ASR attack described in Section 5.3 to SHACAL-2. This leads to the attack on the 37-step reduced SHACAL-2.

Due to a generalized Feistel network-like structure of SHACAL-2, the i-step 32-bit word A_i is computed without using subkeys $W_i, W_{i+1}, ..., W_{i+6}$ as mentioned in [81]. Thus, the 15-step state A_{15} can be computed by each of $\mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$ and $\mathcal{F}^{-1}_{(2)}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(1)} \in \{W_0, W_1, ..., W_{14}\}$ and $\mathcal{K}_{(2)} \in \{W_{22}, ..., W_{36}\}$. Since $|\mathcal{K}_{(1)}| = |\mathcal{K}_{(2)}| = 480 (= 32 \times 15)$ and the size of the matching state A_{15} is 32 bits, by using 22 known plaintext/ciphertext pairs, the time complexity to compute all subkey bits is estimated as

$$C_{comp} = \max(2^{480}, 2^{480}) \times 22 + 2^{37 \cdot 32 - 22 \cdot 32} \approx 2^{485}.$$

The required memory is about 2^{485} blocks. Note that this attack finds a secret key more efficiently than the exhaustive key search only when the key size is more than 485 bits.

Surprisingly, this simple attack exceeds the previous best attack on the reduced SHACAL-2 in the single-key setting for 32 steps [96] with respect to the number of attacked rounds [3]. In the following, by deeply analyzing the functions used in SHACAL-2, we show further improvements.

5.5.3 Advanced ASR Attack on 41-Step Reduced SHACAL-2

In order to extend the basic ASR attack, we choose the lower 4 bits of A_{16} as the matching state. Using this matching state, the 41-step reduced SHACAL-2 can be attacked. The forward and backward functions $\mathcal{F}_{(1)}$ and $\mathcal{F}_{(2)}$ are given as follows (see also Fig. 11.1).

[3]The MITM preimage attacks on the reduced SHA-256 were proposed in [84, 73]. However, these attacks can not be directly applied to SHACAL-2, because in the block cipher setting, these attacks require the code book, i.e., they require all plaintext/ciphertext pairs. Also, due to those high time complexity, they do not seem to work faster than the exhaustive key search.

Forward Computation in $\mathcal{F}_{(1)}$:

Due to the structure of SHACAL-2, the lower 4 bits of A_{16} can be computed from the 15-th state p_{15} and the lower 4 bits of W_{15}, since the other bits of W_{15} are not affected to the lower 4 bits of A_{16}. Thus, the matching state S (the lower 4 bits of A_{16}) is calculated as $S = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{W_0, W_1, ..., W_{14}$, the lower 4 bits of $W_{15}\}$ and $|\mathcal{K}_{(1)}| = 484(= 32 \times 15 + 4)$.

Backward Computation in $\mathcal{F}_{(2)}$:

We give the following observation.

Observation 1. *The lower t bits of A_{j-10} are obtained from the j-th state p_j and the lower t bits of three subkeys W_{j-1}, W_{j-2} and W_{j-3}.*

In the backward computation, i.e., the inverse step function, the $(j-1)$-th step state p_{j-1} is computed from the j-th step state p_j as follows:

$$\begin{aligned} H_{j-1} &= A_j \boxminus \Sigma_0(B_j) \boxminus Maj(B_j, C_j, D_j) \boxminus \Sigma_1(F_j) \\ &\qquad\qquad \boxminus Ch(F_j, G_j, H_j) \boxminus K_{j-1} \boxminus W_{j-1}, \\ D_{j-1} &= E_j \boxminus A_j \boxplus \Sigma_0(B_j) \boxplus Maj(B_j, C_j, D_j), \\ G_{j-1} &= H_j, F_{j-1} = G_j, E_{j-1} = F_j, C_{j-1} = D_j, B_{j-1} = C_j, A_{j-1} = B_j. \end{aligned}$$

Thus, p_{j-1} except for H_{j-1} is obtained from p_j. The lower t bits of H_{j-1} can be computed from p_j and the lower t bits of W_{j-1}. Similarly, from $A_{j-1}, ..., G_{j-1}$ and the lower t bits of H_{j-1} and W_{j-2}, we can compute $A_{j-2}, ..., F_{j-2}$ and the lower t bits of G_{j-2} and H_{j-2}. Furthermore, $A_{j-3}, ..., E_{j-3}$ and the lower t bits of F_{j-3} and G_{j-3} are computed from $A_{j-2}, ..., F_{j-2}$ and the lower t bits of G_{j-2}, H_{j-2} and W_{j-3}. Again, as mentioned in [81], A_{j-10} is determined by p_{j-3} without $W_{j-10}, ..., W_{j-16}$. This relation can be translated to "the lower t bits of A_{j-10} are determined from only $A_{j-3}, ..., E_{j-3}$ and the lower t bits of F_{j-3}, G_{j-3} and H_{j-3}". Therefore, A_{j-10} can be obtained from p_j and the lower t bits of W_{j-1}, W_{j-2} and W_{j-3}.

From Observation 1, the matching state S (the lower 4 bits of A_{16}) can be computed as $S = \mathcal{F}^{-1}_{(2)}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{W_{26}, ..., W_{40}$, the lower t bits of W_{23}, W_{24} and $W_{25}\}$. Thus, $|\mathcal{K}_{(2)}| = 492(= 32 \times 15 + 4 \times 3)$.

Evaluation.

Recall that the matching state S is the lower 4 bits of A_{16}, $|\mathcal{K}_{(1)}| = 484$, $|\mathcal{K}_{(2)}| = 492$ and $|\mathcal{K}_{(3)}| = 336$ $(= 32\times7+(32-4)\times4)$. Thus, using 244 known plaintext/ciphertext pairs (i.e. $N = 244 \leq (484 + 492)/4$), the time complexity for finding all subkeys is estimated as

$$C_{comp} = \max(2^{484}, 2^{492}) \times 244 + 2^{1312-244\cdot4} = 2^{500}.$$

The number of required data is 244 (=max$(244, \lceil(1312 - 244 \cdot 4)/256\rceil)$) known plaintext/ciphertext pairs. The memory size is 2^{492} (=min$(2^{484}, 2^{492}) \times 244$) blocks. The attack works more efficiently than the exhaustive key search when the key size is more than 500 bits.

Table 5.3: Parameters of KATAN family

Algorithm	$\|L_1\|$	$\|L_2\|$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
KATAN32	13	19	12	7	8	5	3	18	7	12	10	8	3
KATAN48	19	29	18	12	15	7	6	28	19	21	13	15	6
KATAN64	25	39	24	15	20	11	9	38	25	33	21	14	9

5.6 Application to KATAN family

In this section, we analyze KATAN family including KATAN32, KATAN48 and KATAN64. First, we briefly describe the specification of KATAN family. Then, we explain our attack strategy for finding the matching state depending on fewer key bits. Finally, we develop ASR attacks on the reduced KATAN32/48/64. We emphasize that all of our attacks on the reduced KATAN presented in this section are the best (exponential-advantage) attacks in the single key setting with respect to the number of attacked rounds.

5.6.1 Description of KATAN

KATAN [24] family is a feedback shift register-based block cipher consisting of three variants: KATAN32, KATAN48 and KATAN64 whose block sizes are 32-, 48- and 64-bit, respectively. All of the KATAN ciphers use the same key schedule accepting an 80-bit key and 254 rounds. The plaintext is loaded into two shift registers L_1 and L_2. Each round, L_1 and L_2 are shifted by one bit, and the least significant bits of L_1 and L_2 are updated by $f_b(L_2)$ and $f_a(L_1)$, respectively. The bit functions f_a and f_b are defined as follows:

$$\begin{aligned} f_a(L_1) &= L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR) \oplus k_{2i}, \\ f_b(L_2) &= L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_{2i+1}, \end{aligned}$$

where $L[x]$ denotes the x-th bit of L, IR denotes the round constant, and k_{2i} and k_{2i+1} denote the 2-bit i-th round key. Note that for KATAN family, the round number starts from 0 instead of 1, i.e., KATAN family consists of round functions starting from the 0-th round to the 253-th round. L_1^i or L_2^i denote the i-th round registers L_1 or L_2, respectively. For KATAN48 or KATAN64, in each round, the above procedure is iterated twice or three times, respectively. All of the parameters for the KATAN ciphers are listed in Table 6.2.

The key schedule of KATAN ciphers copies the 80-bit user-provided key to $k_0, ..., k_{79}$, where $k_i \in \{0, 1\}$. Then, the remaining 428 bits of the round keys are generated as follows:

$$k_i = k_{i-80} \oplus k_{i-61} \oplus k_{i-50} \oplus k_{i-13} \quad \text{for } i = 80, ..., 507.$$

5.6.2 Attack Strategy

Recall that the key of our attack is to find the state that contains as small number of subkey bits as possible. In order to find such states, we exhaustively observe the number of key bits involved in each state per round. A pseudo code for counting the

Algorithm 1 : Counting the number of key bits involved in each state

```
Require: $R$ /* Evaluated number of rounds */
Ensure: $\mathcal{LK}_1[0], \ldots, \mathcal{LK}_1[|L_1|-1]$ and $\mathcal{LK}_2[0], \ldots, \mathcal{LK}_2[|L_2|-1]$ /* The number of
    key bits involved in each state after $R$ round */
1:  $\mathcal{LK}_1[i] \leftarrow 0$ for $i = 0, \ldots, |L_1|-1$
2:  $\mathcal{LK}_2[i] \leftarrow 0$ for $i = 0, \ldots, |L_2|-1$
3:  for $i = 0$ to $R-1$ do
4:    for $j = 0$ to $|L_1|-2$ do
5:      $\mathcal{LK}_1[(|L_1|-1)-j] \leftarrow \mathcal{LK}_1[(|L_1|-1)-j-1]$
6:    end for
7:    for $j = 0$ to $|L_2|-2$ do
8:      $\mathcal{LK}_2[(|L_2|-1)-j] \leftarrow \mathcal{LK}_2[(|L_2|-1)-j-1]$
9:    end for
10:   $\mathcal{LK}_1[0] \leftarrow \mathcal{LK}_2[y_1] + \mathcal{LK}_2[y_2] + \mathcal{LK}_2[y_3] + \mathcal{LK}_2[y_4] + \mathcal{LK}_2[y_5] + \mathcal{LK}_2[y_6] + 1$
11:   $\mathcal{LK}_2[0] \leftarrow \mathcal{LK}_1[x_1] + \mathcal{LK}_1[x_2] + \mathcal{LK}_1[x_3] + \mathcal{LK}_1[x_4] + (\mathcal{LK}_1[x_5] \cdot IR) + 1$
12: end for
13: return $\mathcal{LK}_1[0], \ldots, \mathcal{LK}_1[|L_1|-1]$ and $\mathcal{LK}_2[0], \ldots, \mathcal{LK}_2[|L_2|-1]$
```

number of subkey bits involved in the forward direction for KATAN32 is described in Algorithm 1. We use similar code to observe how many subkey bits are affected to each state of KATAN32 in the backward direction. Also, similar codes are used for counting the number of subkey bits related to each state of KATAN48 and KATAN64. As an example, Table 5.4 shows the results obtained by this algorithm when $R = 63$ of KATAN32 in the forward direction.

5.6.3 ASR Attack on 110-Round Reduced KATAN32

We consider the 110-round variant of KATAN32 starting from the first (0-th) round. In this attack, $L_2^{63}[18]$ is chosen as the matching state.

Forward Computation in $\mathcal{F}_{(1)}$:

As shown in Table 5.4, $L_2^{63}[18]$ depends on 68 subkey bits. This implies that $L_2^{63}[18]$ can be computed by a plaintext P and 68 bits of subkeys. More specifically, $L_2^{63}[18] = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{k_0, ..., k_{54}, k_{56}, k_{57}, k_{58}, k_{60}, ..., k_{64}, k_{68}, k_{71}, k_{73}, k_{77}, k_{88}\}$ and $|\mathcal{K}_{(1)}| = 68$.

Backward Computation in $\mathcal{F}_{(2)}$:

Table 5.5 shows the result obtained by Algorithm 1 modified to backward direction on KATAN32 with $R = 47$ starting from 110 round. In the backward computation, the matching state $L_2^{63}[18]$ is computed as $L_2^{63}[18] = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{k_{126}, k_{138}, k_{142}, k_{146}, k_{148}, k_{150}, k_{153}, k_{154}, k_{156}, k_{158}, k_{160}, \ldots k_{219}\}$, and $|\mathcal{K}_{(2)}| = 70$.

Evaluation.

For the 110-round reduced KATAN32, the matching state S is chosen as $L_2^{63}[18]$ (1-bit state). Since $|\mathcal{K}_{(3)}| = 82(= 2 \times 110 - 68 - 70)$ which is more than 80 bits, we

Table 5.4: Results on KATAN32 with $R = 63$ in forward direction (starting round $= 0$)

$\mathcal{LK}_1[i]$	0	1	2	3	4	5	6	7	8	9	10	11	12						
# key bits	108	104	102	100	98	98	96	94	92	88	86	84	84						
$\mathcal{LK}_2[i]$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**18**
# key bits	104	102	100	98	100	93	92	90	88	90	87	85	83	77	75	75	75	74	**68**

Table 5.5: Results on KATAN32 with $R = 47$ in backward direction (starting round $= 109$)

$\mathcal{LK}_1[i]$	0	1	2	3	4	5	6	7	8	9	10	11	12						
# key bits	44	46	48	50	52	54	56	58	60	62	64	66	68						
$\mathcal{LK}_2[i]$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**18**
# key bits	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62	64	66	68	**70**

first determine only $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$. After that, we additionally mount the MITM approach in order to determine the remaining 82 bits of subkeys.

When $N = 138$ ($\leq (68 + 70)/1$), the time complexity for finding $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ is estimated as

$$C_{comp} = \max(2^{68}, 2^{70}) \times 138 + 2^{138-138\cdot 1} = 2^{77.1}.$$

The number of required data is 138 (=max(138, $\lceil(138 - 138 \cdot 1)/32\rceil$)) known plaintext/ciphertext pairs. The required memory size is about $2^{75.1}$ (=min(2^{68}, 2^{70}) $\times 138$) blocks.

Finally, we need to find the remaining 82 bits of subkeys by using the simple MITM approach in the setting where $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are known. The required complexity and memory for this process is roughly estimated as 2^{41}. These costs are obviously much less than those of finding $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$.

5.6.4 ASR Attack on 100-Round Reduced KATAN48

We consider the 100-round variant of KATAN48 starting from the first (0-th) round. In this attack, $L_2^{58}[28]$ is chosen as the matching state.

In the forward computation, $L_2^{58}[28]$ depends on 71 key bits, namely $L_2^{58}[28] = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{k_0, \ldots k_{60}, k_{62}, k_{64}, k_{65}, k_{66}, k_{69}, k_{71}, k_{72}, k_{75}, k_{79}, k_{86}\}$, and $|\mathcal{K}_{(1)}| = 71$. In the backward computation, $L_2^{58}[28]$ depends on 71 key bits, namely $L_2^{58}[28] = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{k_{116}, k_{122}, k_{124}, k_{128}, k_{130}, k_{132}, k_{134}, k_{135}, k_{136}, k_{138}, \ldots k_{199}\}$, and $|\mathcal{K}_{(2)}| = 71$. Since the size of matching state s is 1, $|\mathcal{K}_{(1)}| = |\mathcal{K}_{(2)}| = 71$ and $|\mathcal{K}_{(3)}| = 58 (= 2 \times 100 - 71 - 71)$, by using $N = 128$ ($\leq (71 + 71)/1$) known plaintext/ciphertext pairs, the time complexity for finding all subkeys is estimated as

$$C_{comp} = \max(2^{71}, 2^{71}) \times 128 + 2^{200-128\cdot 1} = 2^{78}.$$

The number of required data is 128 (=max(128, $\lceil(200 - 128 \cdot 1)/48\rceil$)) known plaintext/ciphertext pairs. The memory size is 2^{78} (=min($2^{71}, 2^{71}$) $\times$ 128) blocks.

Table 5.6: Summary of the attacks in the single-key setting

algorithm	# attacked rounds	time	memory [block]	data	reference
CAST-128	6	$2^{88.51}$	Not given	$2^{53.96}$ KP	[93]
	7	2^{114}	2^{114}	6 KP	Section 5.4.2
Blowfish*1	4	-	-	2^{21} CP	[97]
Blowfish†	8	-	-	2^{48} CP	[91]
	16	2^{292}	2^{260}	9 KP	Section 5.4.3
Blowfish-8R†	8	2^{160}	2^{131}	5 KP	Section 5.4.3
SHACAL-2	32	$2^{504.2}$	$2^{48.4}$	$2^{43.4}$ CP	[96]
	41	2^{500}	2^{492}	244 KP	Section 5.5
KATAN32	78	2^{76}	Not given	2^{16} CP	[98]
	110	2^{77}	$2^{75.1}$	138 KP	Section 5.6.3
KATAN48	70	2^{78}	Not given	2^{31} CP	[98]
	100	2^{78}	2^{78}	128 KP	Section 5.6.4
KATAN64	68	2^{78}	Not given	2^{32} CP	[98]
	94	$2^{77.68}$	$2^{77.68}$	116 KP	Section 5.6.5
FOX128	5	$2^{205.6}$	Not given	2^{9} CP	[99]
	5	2^{228}	2^{228}	14 KP	Section 5.7

† Known F-function setting.

$*1$ The attacks on the full Blowfish in the weak key setting were presented in [91] and [92]

5.6.5 ASR Attack on 94-Round Reduced KATAN64

Similarly to the attack on the 100-round reduced KATAN48, we consider the 94-round variant of KATAN64 starting from the first (0-th) round. In this attack, $L_2^{54}[38]$ is chosen as the 1-bit matching state.

In the forward computation, $L_2^{54}[38]$ depends on 71 subkey bits, namely $L_2^{54}[38] = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{k_0, \dots k_{61},\ k_{63},\ k_{64},\ k_{65},\ k_{66},\ k_{68},\ k_{69},\ k_{71},\ k_{75},\ k_{82}\}$, and $|\mathcal{K}_{(1)}| = 71$. In the backward computation, $L_2^{54}[38]$ depends on 71 subkey bits, namely $L_2^{54}[38] = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{k_{108},\ k_{110},\ k_{114},\ k_{116},\ k_{118},\ k_{120},\ k_{122},\ k_{124}, \dots k_{187}\}$, and $|\mathcal{K}_{(2)}| = 71$. Since $s = 1$, $|\mathcal{K}_{(1)}| = |\mathcal{K}_{(2)}| = 71$, and $|\mathcal{K}_{(3)}| = 46 (= 2 \times 94 - 71 - 71)$, by using $N = 116$ ($\leq (71 + 71)/1$) known plaintext/ciphertext pairs, the time complexity for finding all subkeys is estimated as

$$C_{comp} = \max(2^{71}, 2^{71}) \times 116 + 2^{188-116\cdot 1} = 2^{77.68}.$$

The number of required data is 116 (=max(116, $\lceil (188 - 116 \cdot 1)/64 \rceil$)) known plaintext/ciphertext pairs. The memory size is $2^{77.68}$ (=min$(2^{71}, 2^{71}) \times 116$) blocks.

5.7 Application to FOX128

We apply the ASR attack to the 5-round reduced FOX128.

5.7.1 Description of FOX128

FOX128 is a variant of FOX family [26] consisting of a 16-round modified Lai-Massey scheme with 128-bit block and 256-bit key. A 128-bit input state at round i is denoted as four 32-bit words $(LL_{i-1} \mid\mid LR_{i-1} \mid\mid RL_{i-1} \mid\mid RR_{i-1})$. The i-th round function updates the input state using the 128-bit i-th round key K_i^{rnd} as follows:

$$\begin{aligned}(LL_i||LR_i) &= (\mathtt{or}(LL_{i-1} \oplus \phi_L)||LR_{i-1} \oplus \phi_L) \\ (RL_i||RR_i) &= (\mathtt{or}(RL_{i-1} \oplus \phi_R)||RR_{i-1} \oplus \phi_R),\end{aligned}$$

where `or` denotes a function converting two 16-bit inputs x_0 and x_1 to x_1 and $(x_0 \oplus x_1)$, and $(\phi_L||\phi_R) = \mathtt{f64}((LL_{i-1} \oplus LR_{i-1})||(RL_{i-1} \oplus RR_{i-1}), K_i^{rnd})$. `f64` consisting of two 8 8-bit S-box layers `sigma8` separated by the 8×8 MDS matrix `mu8` returns a 64-bit data from a 64-bit input X and two 64-bit subkeys LK_i^{rnd} and RK_i^{rnd} as $(\mathtt{sigma8}(\mathtt{mu8}(\mathtt{sigma8}(X \oplus LK_i^{rnd})) \oplus RK_i^{rnd}) \oplus LK_i^{rnd})$. Two 64-bit subkeys LK_i^{rnd} and RK_i^{rnd} are derived from K_i^{rnd} as $K_i^{rnd} = (LK_i^{rnd}||RK_i^{rnd})$.

5.7.2 ASR Attack on 5-Round Reduced FOX128

For the 5-round reduced FOX128, the following one-round keyless linear relation can be exploited for the matching:

$$LL_{i+1} \oplus OR^{-1}(LR_{i+1}) = LL_i \oplus LR_i.$$

If we know LL_2 and LR_2, $LL_2 \oplus OR^{-1}(LR_2)$ can be obtained. Thus, we choose LL_2 and LR_2 as the matching state in the forward computation. The 32-bit states LL_2 and LR_2 are computed from a 128-bit subkey K_1^{rnd}, a 64-bit subkey LK_2^{rnd} and the left most 32 bits of RK_2^{rnd}, i.e., $(LL_2, LR_2) = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{K_1^{rnd}, LK_2^{rnd}$, the left most 32 bits of $RK_2^{rnd}\}$, and $|\mathcal{K}_{(1)}| = 224(= 128 + 64 + 32)$. Similarly, we choose LL_3 and LR_3 as the matching state in the backward computation. Since the similar relation holds in the backward computation, LL_3 and LR_3 are computed as $(LL_3, LR_3) = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{K_5^{rnd}, LK_4^{rnd}$, the left most 32 bits of $RK_4^{rnd}\}$, and $|\mathcal{K}_{(2)}| = 224$. Thus, using the parameter $N = 13$ $(\leq (224 + 224)/32)$, the time complexity for finding all round keys is estimated as

$$C_{comp} = \max(2^{224}, 2^{224}) \times 13 + 2^{640-13\cdot 32} = 2^{228}.$$

The number of required data is only 13 (=max(13, $\lceil(640 - 13 \cdot 32)/64\rceil$)) known plaintext/ciphertext pairs, and required memory is about 2^{228} (=min$(2^{224}, 2^{224}) \times 13$) blocks.

5.8 Discussion

On the ASR attack, an attacker attempts to recover all subkeys instead of the user-provided key, regarding all subkeys as independent variables. Note that, if there is no equivalent key, which is a reasonable assumption for a moderate block cipher, there obviously exist unique subkeys that map a given plaintext to the ciphertext encrypted by a secret key. In the standard MITM attack, determining neutral key bits and constructing initial structure called bicliques seem two of the most complicated and important parts in the attack process. However, in our attack, those two procedures are not required, since the attacker focuses only on subkeys and all subkeys are treated equally. Moreover, in the ASR attack, it is not mandatory to analyze the underlying key schedule, since it basically focuses only on the data processing part. These features make the attack simple and generic. While the ASR attack is simple and generic as explained, it is still powerful attack. Indeed, we can significantly improve the previous results on several block ciphers as summarized in Table 10.1 (see also Table 5.2 for the details of the target ciphers). We emphasize that our approach is not polynomial-advantage attack, which requires access of all possible keys, but exponential-advantage attack. Moreover, our attack works on the block ciphers using any key schedule functions even if it is ideal.

While all subkeys are regarded as independent variables in the ASR attack, there must exist some relations between them in an actual block cipher. Thus, if an attacker exploits some properties in the underlying key schedule, the attacker may be able to enhance the attacks presented in this chapter. For instance, the following techniques might be useful:

- Finding the user-provided key from the part of subkeys.
- Reducing the search space of subkeys by using relation of subkeys.

Since the purpose of this chapter provides generic approach to evaluate the security of block ciphers, we do not show these specific and dedicated techniques. However, by using such techniques, the number of attacked rounds might be increased.

5.9 Conclusion

We have proposed a new but simple and generic attack on block ciphers. The proposed attack called all subkeys recovery (ASR) attack is based on the meet-in-the-middle (MITM) attack. The previous MITM attack applied to several block ciphers consisting of a simple key schedule such as a permutation based key schedule, since the MITM attack mainly exploits the weakness in the key schedule. However, there have been a few results on the block ciphers having complex key schedule.

In this chapter, we applied the ASR attack to several block ciphers employing complex key schedule, regarding all subkeys as independent variables. We showed the ASR attacks on the 7-, 41-, 110-, 100-, 94- and 5-round reduced CAST-128, SHACAL-2, KATAN32, KATAN48, KATAN64 and FOX128, respectively. Moreover, we presented the ASR attacks on the full Blowfish in the known F-function setting. All of our results except for the attack on the reduced FOX128 significantly improved the previous results with respect to the number of attacked rounds.

Chapter 6

Related-Key Attack on KATAN32/48/64

KATAN/KTANTAN is a family of hardware oriented block ciphers proposed at CHES 2009. Although the KTANTAN family have been broken by a meet-in-the-middle approach, the KATAN family are secure at present. In this chapter, we investigate the KATAN family in the related-key boomerang framework with several techniques. By using an efficient differential characteristics search method, long boomerang distinguishers can be built. Furthermore, the key recovery phase is optimized by exploiting several properties of the round function such as the high linearity of the round function and the slow key diffusion. As a result, we can attack 174, 145 and 130 rounds of KATAN32, KATAN48 and KATAN64, which substantially improve the known best results whose attacked rounds are 120, 103, 94 rounds, respectively. Our attacks are confirmed by various experimental verifications, especially, we give concrete right quartets for KATAN32.

6.1 Introduction

KATAN/KTANTAN is a family of lightweight block ciphers designed for extremely resource-constrained devices such as RFID and sensor nodes [24]. After its publication in CHES 2009, the full-round KTANTAN family was theoretically broken by using a meet-in-the-middle approach [9]. The attack takes advantage of the simple key scheduling algorithm for the KTANTAN family. The complexity of the attack was later improved by using the splice-and-cut technique [76]. Armed with related-key model, KTANTAN family can even be broken in practical time [100]. For the KATAN family where the key is loaded into a register and updated in each round, the meet-in-the-middle approach is not likely to work well as the cases of KTANTAN. In the single-key setting, a conditional differential attack is applied to 78, 70 and 68 rounds of KATAN32, KATAN48 and KATAN64, respectively [98]. These results were further improved by using a variant of the meet-in-the-middle approach to 110, 100 and 94 rounds [101]. Also, a differential-style attack broke the 115-round KATAN32 [102]. Even in the related-key setting, only 120, 103 and 90 rounds for the respective three versions were broken by the conditional differential attack [103]. Given the full 254 rounds, the KATAN family seem to have enough security margin

Table 6.1: Comparison of attacks against KATAN family

Cipher	Attacking Technique	#Rounds	Time	Data	Mem.	Reference
KATAN32	Differential (SK)	78	2^{76}	2^{16} CP	Not given	[98]
	MITM (SK)	110	2^{77}	138 KP	$2^{75.1}$	[101]
	Differential (SK)	115	2^{79}	138 KP	$2^{75.1}$	[102]
	Differential (RK)	120	2^{31}	Practical (**CP**)	Practical	[103]
	Boomerang (RK)	**172**	$\mathbf{2^{76.2}}$	$\mathbf{2^{27.6}}$ **CP**	$\mathbf{2^{26.6}}$	**Ours**
	Boomerang (RK)	**173**	$\mathbf{2^{77.5}}$	$\mathbf{2^{27.6}}$ **CP**	$\mathbf{2^{26.6}}$	**Ours**
	Boomerang (RK)	**174**	$\mathbf{2^{78.8}}$	$\mathbf{2^{27.6}}$ **CP**	$\mathbf{2^{26.6}}$	**Ours**
KATAN48	Differential (SK)	70	2^{78}	2^{31} CP	Not given	[98]
	MITM (SK)	100	2^{78}	128 KP	2^{78}	[101]
	Differential (RK)	103	2^{25}	Practical (**CP**)	Practical	[103]
	Boomerang (RK)	**145**	$\mathbf{2^{78.5}}$	$\mathbf{2^{38.4}}$ **CP**	$\mathbf{2^{37.4}}$	**Ours**
KATAN64	Differential (SK)	68	2^{78}	2^{32} CP	Not given	[98]
	MITM (SK)	94	$2^{77.68}$	116 KP	$2^{77.68}$	[101]
	Differential (RK)	90	2^{27}	Practical (**CP**)	Practical	[103]
	Boomerang (RK)	**130**	$\mathbf{2^{78.1}}$	$\mathbf{2^{53.1}}$ **CP**	$\mathbf{2^{52.1}}$	**Ours**

SK: Single Key, RK: Related Key, KP: Know Plaintext, CP:Chosen Plaintext.

at present.

In this chapter, we further investigate the security of the KATAN family in the related-key boomerang framework. In order to build a long and efficient boomerang distinguisher, we use an efficient differential characteristics search strategy. Generally speaking, this strategy is inspired by observing that there exists 39 consecutive rounds where the related key difference is zero. We call it blank step, and by fixing the starting round of the blank step, we can go backwards and forwards to compute the input and output differences for both E_0 and E_1. Since the key scheduling algorithm is linear for the KATAN family, key difference fixed in E_1 can still be propagated in backwards deterministically. Although a similar strategy was used for conditional differential attacks [103, 103], we optimize it for boomerang-type attacks. In particular, we carefully choose sets of input differences which are likely to produce differential characteristics with very high probability, and then exhaustively search for differential characteristics of each input set. The probability for E_0 can be further controlled by adding conditions in the plaintexts. By taking multiple output differences for E_0 and multiple input differences for E_1 into consideration, we are able to build 140, 119 and 113 rounds related-key boomerang distinguisher for the corresponding three versions. Based on the boomerang distinguisher, we further optimize the key recovery phase by exploiting the property of the round function in order to reduce the complexity as well as increasing the number of attacked rounds. The comparison of the attacks against the KATAN family is summarized in Table 6.1. Our attacks substantially improve previous attacks for all variants, and are confirmed by various experimental verifications, especially, we give the concrete right quartets for KATAN32 which supports the feasibility of the attack.

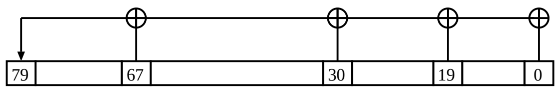


Figure 6.1: Key scheduling function of KATAN32/48/64

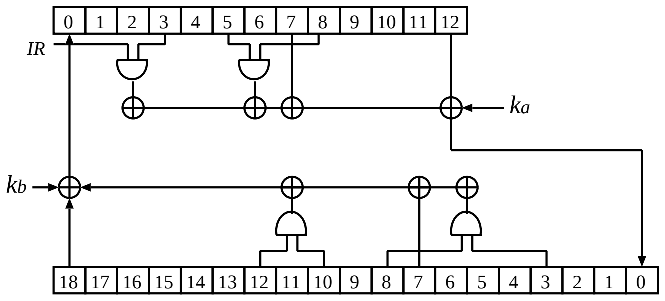


Figure 6.2: Round function of KATAN32

6.2 Preliminaries

6.2.1 KATAN Block Cipher

The KATAN family [24] is a feedback shift register-based block cipher consisting of three variants : KATAN32, KATAN48, KATAN64, whose block sizes are 32 bits, 48 bits and 64 bits, respectively. All variants use the same LFSR(Linear Feedback Shift Register)-type key scheduling function accepting an 80-bit key.

The key scheduling function expands an 80-bit user-provided key k_i $(0 \leq i < 80)$ into a 508-bit subkey sk_i $(0 \leq i < 508)$ by the following linear operations,

$$sk_i = \begin{cases} k_i \ (0 \leq i < 80), \\ k_{i-80} \oplus k_{i-61} \oplus k_{i-50} \oplus k_{i-13} \ (80 \leq i < 508). \end{cases}$$

These operations are expressed as an 80-bit LFSR whose polynomial is $x_{80} + x_{61} + x_{50} + x_{13} + 1$ as shown in Fig 6.1.

In the round function, each bit of a plaintext is loaded into registers L_1 and L_2. Then, these are updated as follows:

$$\begin{aligned} f_a(L_1) &= L_1[x_1] \oplus L_1[x_2] \oplus (L_1[x_3] \cdot L_1[x_4]) \oplus (L_1[x_5] \cdot IR) \oplus k_a, \\ f_b(L_2) &= L_2[y_1] \oplus L_2[y_2] \oplus (L_2[y_3] \cdot L_2[y_4]) \oplus (L_2[y_5] \cdot L_2[y_6]) \oplus k_b, \\ L_1[i] &= L_1[i-1] \ (1 \leq i < |L_1|), \ \ L_1[0] = f_b(L_2), \\ L_2[i] &= L_2[i-1] \ (1 \leq i < |L_2|), \ \ L_2[0] = f_a(L_1), \end{aligned}$$

where $\oplus$ and $\cdot$ are bitwise XOR and AND operations, respectively, and $L[x]$ denotes the x-th bit of L, IR is the round constant value defined in the specification, and k_a and k_b are two subkey bits. Table 6.2 shows the detailed parameters of KATAN32/48/64. For round i, k_a and k_b correspond to $sk_{2(i-1)}$ and $sk_{2(i-1)+1}$, respectively. After 254 rounds (1-254 round), values of registers are output as a ciphertext. Fig. 6.2 illustrates the round function of KATAN32.

Table 6.2: Parameters of KATAN family

Algorithm	$\lvert L_1\rvert$	$\lvert L_2\rvert$	x_1	x_2	x_3	x_4	x_5	y_1	y_2	y_3	y_4	y_5	y_6
KATAN32	13	19	12	7	8	5	3	18	7	12	10	8	3
KATAN48	19	29	18	12	15	7	6	28	19	21	13	15	6
KATAN64	25	39	24	15	20	11	9	38	25	33	21	14	9

6.2.2 Related-Key Boomerang Attack

The related-key boomerang attack [27] is a combination of the boomerang attack [8], and the related-key differential attack [6].

Boomerang-Type Attack

The main idea behind the boomerang attack [8] is to use two short differentials with high probability instead of one long differential with low probability. Suppose that a block cipher with n-bit block and k-bit key, $E : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$, is expressed as a cascade cipher $E = E_1 \circ E_0$, where E_0 has a differential $\alpha \to \beta$ with probability p, and E_1 has a differential $\gamma \to \delta$ with probability q. Then, the distinguisher is mounted as follows:

1 : Ask for the ciphertexts $C_1 = E(P_1)$ and $C_2 = E(P_2)$, where $P_2 = P_1 \oplus \alpha$.

2 : Ask for the plaintexts $P_3 = E^{-1}(C_3)$ and $P_4 = E^{-1}(C_4)$, where $C_3 = C_1 \oplus \delta$ and $C_4 = C_2 \oplus \delta$.

3 : Check whether $P_3 \oplus P_4 = \alpha$.

Here, E satisfies the condition of $P_3 \oplus P_4 = \alpha$ with probability of p^2q^2, while that of a random permutation is 2^{-n}. Note that the attack can be mounted for all possible β's and γ's simultaneously. Therefore, the probability is improved to $\hat{p}^2\hat{q}^2$ from p^2q^2, where $\hat{p} = \sqrt{\sum_\beta Pr^2[\alpha \to \beta]}$ and $\hat{q} = \sqrt{\sum_\gamma Pr^2[\gamma \to \delta]}$.

The amplified boomerang attack converts the adaptive setting into the non-adaptive one [104]. It exploits the birthday paradox in the middle round. An attacker encrypts many plaintext pairs with a difference α, and collects plaintext/ciphertext quartets. Then, she searches for right quartets in the form of $P_1 \oplus P_2 = P_3 \oplus P_4 = \alpha$ and $C_1 \oplus C_3 = C_2 \oplus C_4 = \delta$. For E, this event occurs if the following three conditions are satisfied:

Condition 1 : $E_0(P_1) \oplus E_0(P_2) = E_0(P_3) \oplus E_0(P_4) = \beta$,

Condition 2 : $E_0(P_1) \oplus E_0(P_3)(\text{or } E_0(P_2) \oplus E_0(P_4)) = \gamma$,

Condition 3 : $C_1 \oplus C_3 = C_2 \oplus C_4 = \delta$.

The probability that a quartet is the right one is $2^{-n}p^2q^2$. For a random permutation, this event occurs with probability of 2^{-2n}. Thus, if $pq > 2^{-n/2}$, we can distinguish E from a random permutation. Given N plaintext pairs having α difference, there are $\binom{N}{2} \times 2 \approx N^2$ quartets. Thus, the expected number of right quartets in N pairs is $N^2 \cdot 2^{-n}p^2q^2$. The rectangle attack [105] exploits all β and γ to improve

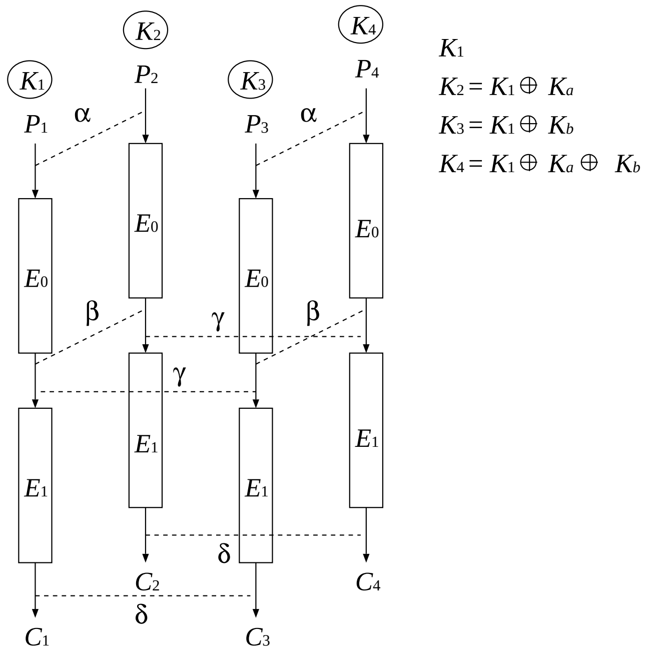


Figure 6.3: Related-key boomerang quartet

the amplified boomerang attack. If $\hat{p}^2\hat{q}^2 > 2^{-n/2}$, this distinguisher works. Though the rectangle attack requires a large amount of data, it can perform a key recovery phase in the non-adaptive setting.

In this chapter, we refer boomerang-type attack using amplified and rectangle techniques to boomerang attack for sake of simplicity.

Related-Key Boomerang Attack

The related-key boomerang attack [27] additionally uses key differences. See Fig. 3 for its illustration. Assume that E_0 has a differential $\alpha \rightarrow \beta$ under a key difference ΔK_a with probability $\hat{p}$, and E_1 has a differential $\gamma \rightarrow \delta$ under a key difference ΔK_b with probability $\hat{q}$. A related-key distinguisher is constructed by using four different unknown keys, K_1, $K_2 = K_1 \oplus K_a$, $K_3 = K_1 \oplus K_b$, $K_3 = K_1 \oplus K_a \oplus K_b$, as follows:

1 : Ask N ciphertext pairs (C_1, C_2), where $C_1 = E_{K_1}(P_1)$, $C_2 = E_{K_2}(P_2)$ and $P_1 \oplus P_2 = \alpha$. Define the set of these pairs as S.

2 : Ask N ciphertext pairs (C_3, C_4), where $C_3 = E_{K_3}(P_3)$, $C_4 = E_{K_4}(P_4)$ and $P_3 \oplus P_4 = \alpha$. Define the set of these pairs as T.

3 : Find right quartets satisfying the following conditions from S and T:

$$P_1 \oplus P_2 = P_3 \oplus P_4 = \alpha \text{ and } C_1 \oplus C_3 = C_2 \oplus C_4 = \delta,$$

Table 6.3: Output differences of z for each input value (x, y) and its difference

Value of	Difference of (x, y)		
(x, y)	(0,1)	(1,0)	(1,1)
(0,0)	0	0	1
(0,1)	0	1	0
(1,0)	1	0	0
(1,1)	1	1	1

Table 6.4: Sets of input differences with Hamming weight 2

set	key difference	no-difference subkeys	plaintext differences	P_{col}
0	0, 19	20 – 98	$L_2[9], L_1[12]$	2^{-2}
1	1, 20	21 – 99	$L_2[18], L_1[2, 7, 12]$	2^{-3}
2	2, 21	22 – 100	$L_2[8], L_1[11]$	2^{-3}
3	3, 22	23 – 101	$L_2[17], L_1[1, 6, 11]$	2^{-3}
4	4, 23	24 – 102	$L_2[7, 18], L_1[10]$	2^{-3}
5	5, 24	25 – 103	$L_2[16], L_1[0, 5, 10]$	2^{-4}
6	6, 25	26 – 104	$L_2[6, 17], L_1[9]$	2^{-3}
7	7, 26	27 – 105	$L_2[15, 18], L_1[4, 9]$	2^{-3}
8	8, 27	28 – 106	$L_2[5, 16], L_1[8]$	2^{-4}
9	9, 28	29 – 107	$L_2[14, 17], L_1[3, 8]$	2^{-5}
10	10, 29	30 – 108	$L_2[4, 15], L_1[7, 12]$	2^{-4}

6.3 Related-Key Boomerang Distinguisher on KATAN32

In this section, we introduce an effective search strategy for finding good related-key differential characteristics. This technique exploits the linearity of the key scheduling and the low dependency of subkey bits. Although a similar search strategy was used in [98, 103], we optimize it for a boomerang-type attack.

6.3.1 Differential Properties of KATAN

Round Function

Let us consider an XOR differential property of the round function of KATAN in which there are four nonlinear components, *i.e.*, AND operations. Table 6.3 shows the differential property of the AND operation whose inputs are x, y and the output is z, namely $z = x \cdot y$. For example, for the value $(1, 0)$ and the difference $(1, 0)$, the difference of z is obtained as $(x \cdot y) \oplus ((x \oplus 1) \cdot (y \oplus 0)) = (1 \cdot 0) \oplus (0 \cdot 1) = 0$. From Table 6.3, when input values have any differences, the output also has a difference with probability $2^{-1}(= 6/12)$.

Besides, one AND operation takes IR as one of the input bits. If one of input bits is public and constant, the corresponding output difference is deterministic. Thus, we can focus on only *three* AND operations as nonlinear operations.

Key Scheduling Function

The key scheduling function employs only linear operations based on the LFSR. Then, we obtain the following observation.

Observation. *Choosing input key differences properly, 79 consecutive subkey bits have no differences after the key scheduling function.*

Since the key scheduling function is the 80-bit LFSR-type construction, any 80 consecutive subkeys surely contain some differences if the key has differences. However, if only one bit of k_i $(0 \leq i \leq 18)$ has a difference, there is no differences in $k_{i+1} - k_{i+79}$, because k_i is not used until k_{i+80}. As for the other case, Table 6.4 shows all possible sets of 2-bit input key differences producing such 79-bit no differences subkeys. For example, assuming that k_0 and k_{19} have differences (set 0), $k_{20} - k_{98}$ do not have differences because the difference k_0 is canceled by k_{19} when it is used for computing k_{80}. The same event occurs in other sets 1-10.

Note that, for all 79 consecutive subkey bits, we can generate the subkey difference which does not make any difference for the target 79 subkey bits. This can be done by the kernel computing approach in [82]. However these sets do not give advantage compared to the sets 1-10, and thus we omit the details.

6.3.2 Strategy for Finding Differential Characteristics

We introduce an effective search strategy for finding good related-key differential characteristics. It is well-suited for boomerang-type attacks in terms of short differential characteristics with very high probability. In general, it is difficult to find good differential characteristics for a bit-oriented cipher due to the large search space. Besides, the related-key setting where key differences are additionally inserted makes it more difficult. In order to get rid of this problem, our strategy is focusing on particular input differential sets which are expected to give good characteristics for boomerang-type attacks.

The differential characteristic search strategy consists of a collision step, a blank step and a brute force step as shown in Fig.6.4.

Collision step : Plaintext difference and key difference cancel each other.

Blank step : No difference exists in registers and inserted subkeys.

Brute force step : Subkey differences propagate to the registers.

The key idea of this strategy is to construct the rounds having no difference called *blank round.* Since the blank round does not reduce the differential probability, *i.e.*, differential probability of such rounds is one, we expect to obtain differential characteristics with high probability. For constructing a long blank round, we utilize the observation 1: *we can set 79 consecutive subkey bits having no difference.* If there is no difference in registers where these 79-bit subkeys are used, the blank round can be easily constructed. In other words, we properly choose plaintext difference for canceling out subkey differences just before the blank round. Table 6.4 shows the plaintext differences for canceling the corresponded input key differences before the blank round and its probability. After the blank round, we search for all differential characteristics. As mentioned before, we regard three AND operations as nonlinear components. Let P_{col}, P_{blk} and P_{bf} be the differential probability of each step, respectively. The whole differential characteristic probability is calculated as $P_{col} \cdot P_{blk} \cdot P_{bf}$, where $P_{blk} = 1$.

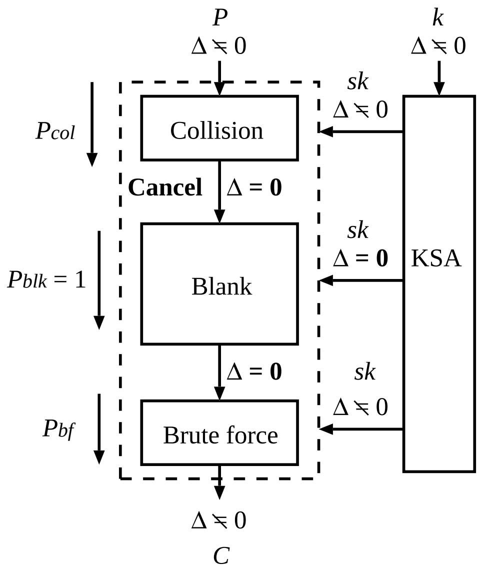


Figure 6.4: Strategy for finding differential characteristics

Table 6.5: Maximum probability of differential characteristics of each set in E_0

Round	**set0**	**set1**	**set2**	**set3**	**set4**	**set5**	**set6**	**set7**	**set8**	**set9**	**set10**
65	2^{-9}	2^{-9}	2^{-7}	2^{-8}	2^{-7}	2^{-8}	2^{-7}	2^{-7}	2^{-7}	2^{-7}	2^{-7}
66	2^{-10}	2^{-10}	2^{-7}	2^{-9}	2^{-7}	2^{-9}	2^{-7}	2^{-8}	2^{-8}	2^{-8}	2^{-8}
67	2^{-12}	2^{-10}	2^{-8}	2^{-10}	2^{-7}	2^{-10}	2^{-7}	2^{-9}	2^{-8}	2^{-9}	2^{-9}
68	2^{-13}	2^{-11}	2^{-9}	2^{-10}	2^{-8}	2^{-11}	2^{-7}	2^{-11}	2^{-8}	2^{-10}	2^{-10}
69	2^{-14}	2^{-12}	2^{-10}	2^{-12}	2^{-9}	2^{-11}	2^{-8}	2^{-12}	2^{-8}	2^{-11}	2^{-11}
70	2^{-15}	2^{-12}	2^{-12}	2^{-12}	2^{-10}	2^{-12}	2^{-9}	2^{-12}	2^{-9}	2^{-12}	2^{-12}
71	2^{-16}	2^{-13}	2^{-13}	2^{-12}	2^{-12}	2^{-13}	2^{-10}	2^{-14}	2^{-10}	2^{-12}	2^{-12}

Input key differences are restricted to the set satisfying the property of the observation 1. Then, plaintext differences are also determined from the set of input key differences for constructing the blank round (see Table 6.4).

6.3.3 Related-key Boomerang Distinguisher on 140-round KATAN32

Using the efficient differential characteristics search, we obtain the maximum probability of differential characteristics of each input set in E_0 starting from round 1 (see Table 6.5).

To construct a distinguisher, we choose 70 rounds of set 8 whose probability is highest of all the sets. E_0 has 8 characteristics with probability $p = 2^{-9}$, 16 characteristics with probability $p = 2^{-10}$, 16 characteristics with probability $p = 2^{-11}$ and 64 characteristics with probability $p = 2^{-12}$, which are generated from the same input. Thus, the overall probability for E_0 is

$$\hat{p} = \sqrt{(2^{-9})^2 \cdot 8 + (2^{-10})^2 \cdot 16 + (2^{-11})^2 \cdot 16 + (2^{-12})^2 \cdot 64} \approx 2^{-7.1}.$$

Table 6.7 gives a single differential trail of E_0 with probability of 2^{-9}, where round

Table 6.6: Maximum probability of differential characteristics of each set in E_1

Round	**set0**	**set1**	**set2**	**set3**	**set4**	**set5**	**set6**	**set7**	**set8**	**set9**	**set10**
65	2^{-6}	2^{-10}	2^{-7}	2^{-8}	2^{-9}	2^{-8}	2^{-7}	2^{-7}	2^{-8}	2^{-7}	2^{-6}
66	2^{-7}	2^{-11}	2^{-7}	2^{-9}	2^{-9}	2^{-9}	2^{-7}	2^{-8}	2^{-9}	2^{-8}	2^{-7}
67	2^{-8}	2^{-11}	2^{-8}	2^{-10}	2^{-10}	2^{-11}	2^{-7}	2^{-9}	2^{-10}	2^{-9}	2^{-8}
68	2^{-9}	2^{-13}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-7}	2^{-10}	2^{-10}	2^{-10}	2^{-8}
69	2^{-11}	2^{-13}	2^{-10}	2^{-12}	2^{-13}	2^{-12}	2^{-8}	2^{-11}	2^{-11}	2^{-11}	2^{-8}
70	2^{-12}	2^{-13}	2^{-12}	2^{-12}	2^{-14}	2^{-13}	2^{-9}	2^{-11}	2^{-12}	2^{-12}	2^{-8}
71	2^{-15}	2^{-14}	2^{-13}	2^{-12}	2^{-15}	2^{-14}	2^{-10}	2^{-12}	2^{-14}	2^{-12}	2^{-9}

0 means initial differences, *i.e.*, differences of a plaintext.

Since KATAN employs the LFSR-based key scheduling, all 508 subkey bits can be calculated from any consecutive 80 subkey bits. It means that we can use the efficient differential characteristics search strategy from any round by regarding the consecutive 80 subkey bits as the master key bits. Thus, we search for differential characteristics of E_1 from round 71 with the same strategy.

Table 6.6 shows the maximum probability of differential characteristics of each set in E_1 starting from round 71. We choose set 1 as E_1 with probability 2^{-8}. In addition, E_1 has 4 characteristics with probability 2^{-8}, 8 characteristics with probability 2^{-9} and 32 characteristics with probability 2^{-10}, which produce the same output difference. Thus, the total probability for E_1 is estimated as

$$\hat{q} = \sqrt{(2^{-8})^2 \cdot 4 + (2^{-9})^2 \cdot 8 + (2^{-10})^2 \cdot 32} \approx 2^{-6.5}.$$

Table 6.8 gives a single differential trail of E_1 with probability 2^{-8}.

Combining these two-type differential characteristics, 140 (=70+70)-round related-key boomerang distinguisher can be constructed with probability of

$$\hat{p}^2 \cdot \hat{q}^2 = (2^{-7.1})^2 \cdot (2^{-6.5})^2 = 2^{-27.2}\ (> 2^{-32}).$$

The probability of the boomerang distinguisher, $2^{-27.2}$, is possible to verify practically. We performed the experiment on a standard PC and found right quartets within a few minutes. One examples is shown in Table 6.9 .

Table 6.9: Example of confirmed boomerang quartets for KATAN32

P_1	`0x46ec3236`	C_1	`0xee39e8a1`	K_1	`0x22fe640869975423bce9`
P_2	`0x4eed3216`	C_2	`0xf19133e1`	K_2	`0x22fe640869975c23bde9`
P_3	`0xd2379460`	C_3	`0xee11e925`	K_3	`0xa6ffe4826d8d3228d6c1`
P_4	`0xda369440`	C_4	`0xf1b93265`	K_4	`0xa6ffe4826d8d3a28d7c1`
$P_1 \oplus P_2$	`0x08010020`	$C_1 \oplus C_3$	`0x00280184`	$K_1 \oplus K_2 = K_3 \oplus K_4$	`0x00000000000008000100`
$P_3 \oplus P_4$	`0x08010020`	$C_2 \oplus C_4$	`0x00280184`	$K_1 \oplus K_3 = K_2 \oplus K_4$	`0x8401808a041a660b6a28`

Table 6.7: Differential characteristic of KATAN32 E_0 (1 - 70)

Round	$L_2[0] \dots L_2[18]$	$L_1[0] \dots L_1[13]$	K_a	K_b	Pr.
0	0000010000000000100	0000000010000	0	0	1
1	0000001000000000010	0000000001000	0	0	2^{-1}
2	0000000100000000001	0000000000100	0	0	2^{-1}
3	0000000010000000000	0000000000010	0	0	2^{-1}
4	0000000001000000000	0000000000001	1	0	2^{-2}
5	0000000000100000000	0000000000000	0	0	2^{-2}
6	0000000000010000000	0000000000000	0	0	2^{-3}
7	0000000000001000000	0000000000000	0	0	2^{-3}
8	0000000000000100000	0000000000000	0	0	2^{-4}
9	0000000000000010000	0000000000000	0	0	2^{-4}
10	0000000000000001000	0000000000000	0	0	2^{-4}
11	0000000000000000100	0000000000000	0	0	2^{-4}
12	0000000000000000010	0000000000000	0	0	2^{-4}
13	0000000000000000001	0000000000000	0	1	2^{-4}
14	0000000000000000000	0000000000000	0	0	2^{-4}
53	0000000000000000000	0000000000000	0	1	2^{-4}
54	0000000000000000000	1000000000000	0	0	2^{-4}
55	0000000000000000000	0100000000000	0	0	2^{-4}
56	0000000000000000000	0010000000000	0	0	2^{-4}
57	0000000000000000000	0001000000000	0	0	2^{-4}
58	0000000000000000000	0000100000000	0	0	2^{-4}
59	0000000000000000000	0000010000000	0	0	2^{-4}
60	0000000000000000000	0000001000000	1	0	2^{-5}
61	1000000000000000000	0000000100000	0	0	2^{-5}
62	1100000000000000000	0000000010000	0	0	2^{-5}
63	0110000000000000000	0000000001000	0	0	2^{-6}
64	0011000000000000000	0000000000100	0	0	2^{-6}
65	0001100000000000000	0000000000010	0	0	2^{-7}
66	0000110000000000000	0000000000001	0	1	2^{-8}
67	1000011000000000000	1000000000000	0	0	2^{-8}
68	0100001100000000000	0100000000000	0	0	2^{-8}
69	0010000110000000000	1010000000000	0	0	2^{-8}
70	0001000011000000000	1101000000000	0	0	2^{-9}

Table 6.8: Differential characteristic of KATAN32 E_1 (71 - 140)

Round	$L_2[0] \dots L_2[18]$	$L_1[0] \dots L_1[13]$	K_a	K_b	Pr.
70	0000100000000001000	0000000100001	0	0	1
71	0000010000000000100	0000000010000	0	0	1
72	0000001000000000010	0000000001000	0	0	2^{-1}
73	0000000100000000001	0000000000100	0	0	2^{-1}
74	0000000010000000000	0000000000010	0	0	2^{-1}
75	0000000001000000000	0000000000001	1	0	2^{-2}
76	0000000000100000000	0000000000000	0	0	2^{-2}
77	0000000000010000000	0000000000000	0	0	2^{-3}
78	0000000000001000000	0000000000000	0	0	2^{-3}
79	0000000000000100000	0000000000000	0	0	2^{-4}
80	0000000000000010000	0000000000000	0	0	2^{-4}
81	0000000000000001000	0000000000000	0	0	2^{-4}
82	0000000000000000100	0000000000000	0	0	2^{-4}
83	0000000000000000010	0000000000000	0	0	2^{-4}
84	0000000000000000001	0000000000000	0	1	2^{-4}
124	0000000000000000000	0000000000000	0	1	2^{-2}
125	0000000000000000000	1000000000000	0	0	2^{-2}
126	0000000000000000000	0100000000000	0	0	2^{-3}
127	0000000000000000000	0010000000000	0	0	2^{-3}
128	0000000000000000000	0001000000000	0	0	2^{-3}
129	0000000000000000000	0000100000000	0	0	2^{-4}
130	0000000000000000000	0000010000000	0	0	2^{-4}
131	0000000000000000000	0000001000000	1	0	2^{-5}
132	1000000000000000000	0000000100000	0	0	2^{-5}
133	1100000000000000000	0000000010000	0	0	2^{-5}
134	0110000000000000000	0000000001000	0	0	2^{-6}
135	0011000000000000000	0000000000100	0	0	2^{-6}
136	0001100000000000000	0000000000010	0	0	2^{-7}
137	0000110000000000000	0000000000001	0	1	2^{-8}
138	1000011000000000000	1000000000000	0	0	2^{-8}
139	0100001100000000000	0100000000000	0	0	2^{-8}
140	0010000110000000000	1010000000000	0	0	2^{-8}

6.4 Related-Key Recovery Attack on KATAN-32

In this section, a related-key attack on KATAN-32 is proposed given the 140-round boomerang distinguisher. One of the challenging problems is how to reduce the candidate quartets. This is usually achieved by studying the propagation of the difference to the ciphertext in order to filter out definitely wrong quartets. For the KATAN family, this may not be the best option. When we extend the attacking rounds as long as possible, the difference propagation will leave us with no clue. Instead, we try to choose plaintext so that the characteristic for the first several rounds are always satisfied. This strategy is also used in previous KATAN attacks [98, 103].

We further optimize the key recovery phase by exploiting the property of the round function, in order to reduce the complexity as well as increasing the number of attacked rounds.

6.4.1 Conditions for Chosen Plaintexts

In the collision step for E_0, we have calculated that $p_{col} = 2^{-4}$. Recall that for two inputs to an AND gate, one input with value 1 will guarantee the propagation of the difference from the other input, and difference will disappear when the value is fixed to 0. Thus we can assure the difference propagation with probability 1 by fixing some of the plaintext bits. For KATAN32, the probability for the collision steps can be increased to 1. The conditions on plaintext bits are $L_2[0] = L_2[3] = L_2[7] = L_1[5] = 0$, and the increased probability for E_0 is

$$\hat{p} = \sqrt{(2^{-9+4})^2 \cdot 8 + (2^{-10+4})^2 \cdot 16 + (2^{-11+4})^2 \cdot 16 + (2^{-12+4})^2 \cdot 64} \approx 2^{-3.1}.$$

This indicates that we can expect one right quartet in $2^{51.2}(= (2^{-3.1})^2 \cdot (2^{-6.5})^2 \cdot 2^{-32})$. As a result, the number of quartet candidates is reduced to $2^{51.2}$.

6.4.2 Optimizing Key Recovery Phase

Suppose that we append x rounds to the end of the 140-round distinguisher. The attacker queries N pairs of plaintexts to oracles with K_1 and K_2. She also queries N pairs of plaintexts to oracles with K_3 and K_4. Then, N^2 quartets are constructed. We set $N \leftarrow \hat{p}^{-1} \cdot \hat{q}^{-1} \cdot 2^{n/2}$ so that a right quartet is generated.

To recover subkeys for the last x rounds with a straight-forward method, the attacker guesses all subkeys for the last x rounds, and performs partial decryptions until the end of the 140-round distinguisher for each of N^2 quartets. Let g_i, where $i \in \{1, 2, 3, 4\}$ be a set of subkey bits used in the last x rounds for the K_i oracle. Because KATAN uses two subkey bits in each round, each g_i contains $2x$ subkey bits. We denote the x-round partial decryption for a ciphertext C_i with a guessed key g_i by $D_{g_i}(C_i)$. Note that if the guess for K_1 oracle, g_1, is determined, the corresponding g_2, g_3, and g_4 are determined uniquely. If the guessed value is correct, the attacker will find one quartet such that $D_{g_1}(C_1) \oplus D_{g_3}(C_3) = D_{g_2}(C_2) \oplus D_{g_4}(C_4) = \delta$. If such a quartet is not found, the guess is wrong. Unfortunately, the complexity of this approach is too high. Let $\#g$ be the number of subkey bits in each of g_i, namely $2x$. The approach requires $N^2 \cdot \#g \cdot 4$ partial decryptions, where a factor of N^2 is too high.

Pairwise Approach

We propose a more efficient method. For each guess of g_1 and corresponding g_2, g_3, g_4, we perform the partial decryption for N pairs of (C_1, C_2) and N pairs of (C_3, C_4) independently, and identify the right quartet by checking their match as follows:

1. Make a guess for g_1 and determine the corresponding values for g_2, g_3, g_4.
2. For all N pairs of (C_1, C_2), compute $(D_{g_1}(C_1) \oplus \delta, D_{g_2}(C_2) \oplus \delta)$ and store them in a table with N entries.

Table 6.10: Partial-matching technique for KATAN32

#Skipped rounds	Number of bits with unknown differences			P_{right}
	L_1	L_2	Total	
1–4	0	0	0	$N^2 \cdot 2^{-64}$
5	1	0	1	$N^2 \cdot 2^{-62}$
6	2	0	2	$N^2 \cdot 2^{-60}$
7	3	1	4	$N^2 \cdot 2^{-56}$
$r(\geq 6)$	$r-4$	$r-6$	$2r-10$	$N^2 \cdot 2^{-84+4r}$

If one subkey bit for the first skipped round is guessed, P_{right} decreases by 2^2.

3. For all N pairs of (C_3, C_4), compute $(D_{g_3}(C_3), D_{g_4}(C_4))$ and store them in another table.
4. If the guess is correct, a match is found. Otherwise, the guess is discarded.

This method requires only $N \cdot \#g \cdot 2$ partial decryptions for Step 2 and Step 3 respectively, in total $N \cdot \#g \cdot 4$ partial decryptions. The memory requirement is $2N$ state. The memory for Step 3 can be saved by checking the match as soon as we obtain a pair. Each guess is judged as a right-key candidate if one of N^2 quartets satisfies two n-bit relations δ. We denote this probability by P_{right}, which is $N^2 \cdot 2^{-2n}$. After the analysis, the key space will be $\#g \cdot P_{right} = \#g \cdot N^2 \cdot 2^{-2n}$. The remaining key space will be later examined by the exhaustive search.

Exploiting Linear Subkey Insertion

We further optimize the attack by exploiting the round function structure. Recall Fig. 6.2. If the output *value* for some round r is known, the input *difference* for round r can be computed without guessing subkeys. This is because the 1-round decryption uses subkey values only in the linear operation. The situation continues until unknown values are used as an input of AND operations. In the end, the difference after the x-round decryption can be computed only with guessing subkeys for the last $x-4$ rounds.

Partial Matching

Another optimization is possible by exploiting the property that only 2 bits are updated in each decryption round. Let us see what will happen if we go back 5 rounds without guessing subkeys. As mentioned above, the difference in all bits can be computed up to 4 rounds. In the next round, the attacker cannot compute the difference of the updated bit $L_1[12]$, while she knows the difference of the other 31 bits ($L_2[18]$ can be computed at this stage). Hence, the match can be performed for 31 bits. The analysis is summarized in Table 6.10. Let r be the number of rounds which we compute without guessing subkeys. Let z be the number of bits with unknown difference. The match is performed for $32-z$ bits. From Table 6.10, $z = 2r - 10$ for $r \geq 6$. Because 2 pairs exist in a quartet, P_{right} is $N^2 \cdot 2^{-2(32-z)}$, which is $N^2 \cdot 2^{-84+4r}$. As long as P_{right} is small enough, subkeys can be recovered faster than the exhaustive search.

Partial Key Guessing

The last technique for the optimization is partially guessing a subkey, *i.e.*, only guessing 1 bit of a subkey in the first skipped rounds. In Table 6.10, this makes the number of unknown bits be $2r - 11$ and P_{right} be $N^2 \cdot 2^{-86+4r}$ when $r \geq 6$. Intuitively, the technique increases the computational complexity by 1 bit due to the additional guessed bit, while it increases the efficiency of the filtering function by 2 bits due to two pairs in a quartet.

6.4.3 Attack Procedure and Complexity Evaluation

We append $x = 34$ rounds to the end of the 140-round distinguisher. The number of rounds which we do not guess subkey values, r, is 8, but we use the partial key guessing technique. Therefore, $\#g = 53$, where each g_i consists of 52 bits for the last 26 rounds and 1 bit of subkey (either bit is fine) for the 27th last round.

1. Choose $N = 2^{25.6}$ plaintext pairs (P_1, P_2) so that $P_1 \oplus P_2 = \alpha$ and satisfy the 4-bit conditions $L_2[0] = L_2[3] = L_2[7] = L_1[5] = 0$. Query them to the oracles with K_1 and K_2, and store the corresponding $2^{25.6}$ pairs of (C_1, C_2).

2. Do the same for (P_3, P_4) to obtain $N = 2^{25.6}$ ciphertext pairs (C_3, C_4).

3. Guess g_1 and the corresponding g_2, g_3, g_4. For each guess, do as follows.
   (a) For $2^{25.6}$ pairs of (C_1, C_2), decrypt them for 26 rounds. Then, further decrypt them by 8 rounds to obtain differences in $32 - (2 \times 8 - 11) = 27$ bits, and take the XOR with δ. Store them in a table with $2^{25.6}$ entries.
   (b) For $2^{25.6}$ pairs of (C_3, C_4), do as follows.
      i. Similarly decrypt the pair for $26 + 8 = 34$ rounds to obtain the differences in 27 bits.
      ii. Check if the match exists between the stored values. If no match is found, delete the guess from the candidate. Otherwise, do as follows.
      iii. For exhaustive guesses of $80 - 53 = 27$-bit subkeys which are not guessed yet, check the correctness of the guess by using any pair of plaintext and ciphertext (32-bit match). It it passes the check, then further check the correctness of the guess with two more plaintext-ciphertext pairs. If it passes all checks, output it as the correct key.

For Step 1 and 2, we need $4 * 2^{25.6} = 2^{27.6}$ chosen plaintexts. Step 3a requires $2^{53+25.6} \cdot 2 \cdot 34/174 \approx 2^{77.25}$ 174-round KATAN32 computations. The memory requirement for Step 3a is about $2 \cdot 2^{25.6} = 2^{26.6}$ state values. Step 3(b)i also requires $2^{77.25}$ computations. After Step 3(b)ii, $2^{53} \cdot P_{right} = 2^{53} \cdot (2^{51.2} \cdot 2^{-86+4 \cdot 8}) = 2^{50.2}$ key candidates will remain. Step 3(b)iii requires $2^{50.2+27} = 2^{77.2}$ 174-round KATAN32 computations for the first plaintext-ciphertext pair. Only $2^{77.2-32} = 2^{45.2}$ key candidates are examined for the second pair, and only $2^{45.2-32} = 2^{13.2}$ candidates are examined for the third pair. Hence, the complexity for Step 3(b)iii is $2^{77.2}+2^{45.2}+2^{13.2} \approx 2^{77.2}$ 174-round KATAN32 computations.

In summary, the data complexity is $2^{27.6}$ chosen plaintexts, the time complexity is $2^{77.25} + 2^{77.25} + 2^{77.2} \approx 2^{78.8}$ 174-round KATAN32 computations. The memory requirement is $2^{25.6}$ state.

Note that our attack succeeds only if the right quartet is obtained *i.e.*, the differential with a probability of $2^{-51.2}$ is satisfied with $2^{51.2}$ quartets. Hence, the success probability of our attack is $1 - 1/e \approx 0.63$. On the other hand, the success probability of the brute force attack with $2^{78.8}$ trials is 0.44. Hence, our attack is better than the brute force attack with the same complexity.

Also note that the advantage of our attack becomes clearer if the number of rounds is reduced more. For example, the complexity for 173 or 172 rounds is $2^{77.5}$ or $2^{76.2}$ computations, respectively, with the same data and memory.

6.5 Related-Key Boomerang Attack on KATAN-48/64

6.5.1 Differential Characteristics and Plaintext Conditions

First we give differential characteristic for KATAN48. Similar to KATAN32, we start from finding collision steps, and by changing the starting point of the collision steps, we go backwards to derive the input differences and key differences. As a result we build a 119-round boomerang distinguisher for KATAN48. Table 6.11 and 6.12 demonstrate one characteristic for E_0 and E_1. We use a fixed characteristic between rounds 1 to 49 of E_0 and rounds 70 to 119 for E_1, while we use a differential for the other rounds. In total, for E_0 there are 32 characteristics with probability 2^{-14}, 128 characteristics with probability 2^{-15} and 128 characteristics with probability 2^{-16}. For E_1 there are 128 characteristics with probability 2^{-12}. As a result, $\hat{p} = \sqrt{(2^{-14})^2 \cdot 32 + (2^{-15})^2 \cdot 128 + (2^{-16})^2 \cdot 128} = 2^{-10.9}$, $\hat{q} = \sqrt{(2^{-12})^2 \cdot 128} = 2^{-8.5}$.

Differential characteristics for E_0 and E_1 of KATAN64 are summarized in Table 6.13 and 6.14. Due to the more scrambling in each round, the number of the collision steps and the brute force steps are reduced. We use a fixed characteristic between rounds 1 to 46 of E_0 and rounds 103 to 113 for E_1, while we use a differential for the other rounds. For E_0 there are 64, 256, 512, 1024, and 1024 characteristics with probability 2^{-16}, 2^{-17}, 2^{-18}, 2^{-19}, and 2^{-20}, respectively. For E_1 there are 4, 24, 88, 224, 416, 608, 704, 640, and 256 characteristics with probability 2^{-16}, 2^{-17}, 2^{-18}, 2^{-19}, 2^{-20}, 2^{-21}, 2^{-22}, 2^{-23}, and 2^{-24}, respectively. As a result, $\hat{p} = 2^{-12.25}$, and $\hat{q} = 2^{-13.8}$.

The probabilities of the collision steps of E_0 for KATAN48/64 are both 2^{-7}, but this can be improved by 2^7 by choosing the plaintext satisfying the conditions. The conditions are given below along with the improved probability for $\hat{p}$. $\hat{q}$ is not affected by the chosen plaintext.

KATAN48

Conditions: $L_2[0] = L_2[1] = L_2[2] = L_2[11] = L_2[17] = 0, L_2[10] \neq L_2[18]$.
$\hat{p} = \sqrt{(2^{-14+7})^2 \cdot 32 + (2^{-15+7})^2 \cdot 128 + (2^{-16+7})^2 \cdot 128} = 2^{-3.9}$. We expect $2^{72.8}(= (2^{3.9})^2 \cdot (2^{8.5})^2 \cdot 2^{48})$ quartets before a right one shows up.

Table 6.11: Differential characteristic of KATAN48 E_0 (1 - 60)

Round	$L_2(L_2[0]...L_2[28])$	$L_1(L_1[0]...L_1[18])$	K_a	K_b	Pr.
0	00000000011000000011000000011	0000000000000000011	1	0	1
1	00000000000110000000110000000	0000000000000000000	0	0	1
2	00000000000001100000001100000	0000000000000000000	0	0	2^{-2}
3	00000000000000011000000011000	0000000000000000000	0	0	2^{-4}
4	00000000000000000110000000110	0000000000000000000	0	0	2^{-5}
5	00000000000000000001100000001	0000000000000000000	0	0	2^{-5}
6	00000000000000000000011000000	0000000000000000000	0	0	2^{-6}
7	00000000000000000000000110000	0000000000000000000	0	0	2^{-7}
8	00000000000000000000000001100	0000000000000000000	0	0	2^{-7}
9	00000000000000000000000000011	0000000000000000000	0	1	2^{-7}
10	00000000000000000000000000000	0000000000000000000	0	0	2^{-7}
49	00000000000000000000000000000	0000000000000000000	0	1	2^{-7}
50	00000000000000000000000000000	1100000000000000000	0	0	2^{-7}
51	00000000000000000000000000000	0011000000000000000	0	0	2^{-7}
52	00000000000000000000000000000	0000110000000000000	0	0	2^{-7}
53	10000000000000000000000000000	0000001100000000000	0	0	2^{-7}
54	00100000000000000000000000000	0000000011000000000	0	0	2^{-9}
55	00001000000000000000000000000	0000000000110000000	0	0	2^{-9}
56	10000010000000000000000000000	0000000000001100000	1	0	2^{-9}
57	10100000100000000000000000000	0000000000000011000	0	0	2^{-10}
58	00101000001000000000000000000	0000000000000000110	0	0	2^{-12}
59	10001010000010000000000000000	0000000000000000001	0	0	2^{-12}
60	01100010100000100000000000000	0000000000000000000	0	0	2^{-14}

KATAN64

Conditions: $L_2[6] = L_2[7] = L_2[8] = L_2[21] = L_2[30] = 0, L_2[20] \neq L_2[32], L_2[19] \neq L_2[31]$. $\hat{p}$ becomes $2^{-5.25}$. We expect $2^{102.1}(= (2^{5.25})^2 \cdot (2^{13.8})^2 \cdot 2^{64})$ quartets before a right one shows up.

6.5.2 Optimization and Summary of Key Recovery Attacks

The overall strategy is the same as the one for KATAN32. The only difference from KATAN32 is the impact of the partial-matching technique, which comes from the different register sizes $|L_1|, |L_2|$ and input-bit positions for AND operations. The results are summarized in Table 6.15 and Table 6.16.

145-round KATAN48

The attack generates $\hat{p}^{-1} \cdot \hat{q}^{-1} \cdot 2^{48/2} = 2^{3.9+8.5+24} = 2^{36.4}$ pairs of (P_1, P_2), and $2^{36.4}$ pairs of (P_3, P_4). This makes $2^{72.8}$ quartets, which include a right quartet with probability 0.63. We append 26 rounds after the 119-round distinguisher. Hence, 145 rounds are attacked. In the key recover phase, we guess 42 bits of subkeys for the last 21 rounds. Therefore, the number of skipped steps, r, is 5. This makes

Table 6.12: Differential characteristic of KATAN48 E_1 (61 - 119)

Round	$L_2(L_2[0]...L_2[28])$	$L_1(L_1[0]...L_1[18])$	K_a	K_b	Pr.
60	00000000011000000011000000011	0000000000000000011	1	0	1
61	00000000000110000000110000000	0000000000000000000	0	0	1
62	00000000000001100000001100000	0000000000000000000	0	0	2^{-2}
63	00000000000000011000000011000	0000000000000000000	0	0	2^{-4}
64	00000000000000000110000000110	0000000000000000000	0	0	2^{-5}
65	00000000000000000001100000001	0000000000000000000	0	0	2^{-5}
66	00000000000000000000011000000	0000000000000000000	0	0	2^{-6}
67	00000000000000000000000110000	0000000000000000000	0	0	2^{-7}
68	00000000000000000000000001100	0000000000000000000	0	0	2^{-7}
69	00000000000000000000000000011	0000000000000000000	0	1	2^{-7}
70	00000000000000000000000000000	0000000000000000000	0	0	2^{-7}
109	00000000000000000000000000000	0000000000000000000	0	1	2^{-7}
110	00000000000000000000000000000	1100000000000000000	0	0	2^{-7}
111	00000000000000000000000000000	0011000000000000000	0	0	2^{-7}
112	00000000000000000000000000000	0000110000000000000	0	0	2^{-7}
113	10000000000000000000000000000	0000001100000000000	0	0	2^{-7}
114	00100000000000000000000000000	0000000011000000000	0	0	2^{-9}
115	00001000000000000000000000000	0000000000110000000	0	0	2^{-9}
116	10000010000000000000000000000	0000000000001100000	1	0	2^{9}
117	10100000100000000000000000000	0000000000000011000	0	0	2^{-10}
118	00101000001000000000000000000	0000000000000000110	0	0	2^{-12}
119	10001010000010000000000000000	0000000000000000001	0	0	2^{-12}

the time complexity for the analysis for P_1, P_2 pairs be $2^{36.4+42} \cdot 2 \cdot 26/145 \approx 2^{76.9}$ 145-round KATAN48 computations. The memory requirement is $2 \cdot 2^{36.4} = 2^{37.4}$ state values. The analysis for P_3, P_4 pairs also requires $2^{76.9}$ 145-round KATAN48 computations. P_{right} is $2^{72.8} \cdot 2^{-76} = 2^{-3.2}$. Hence, the complexity for the exhaustive check becomes $2^{80} \cdot P_{right} = 2^{76.8}$. In the end, the data complexity is $4 \cdot 2^{36.4} = 2^{38.4}$ chosen plaintexts. The computational complexity is $2^{76.9} + 2^{76.9} + 2^{76.8} \approx 2^{78.5}$ 145-round KATAN48 computations. The success probability of our attack is 0.63, while the success probability of the brute force attack with the same complexity is 0.35.

130-round KATAN64

The attack generates $\hat{p}^{-1} \cdot \hat{q}^{-1} \cdot 2^{64/2} = 2^{5.25+13.8+32} = 2^{51.05}$ pairs of (P_1, P_2), and $2^{51.05}$ pairs of (P_3, P_4). This makes $2^{102.1}$ quartets, which include a right quartet with probability 0.63. We append 17 rounds after the 113-round distinguisher. Hence, 130 rounds are attacked. In the key recover phase, we guess 28 bits of subkeys for the last 14 rounds. Therefore, the number of skipped steps, r, is 3. This makes the time complexity for the analysis for P_1, P_2 pairs be $2^{51.05+28} \cdot 2 \cdot 17/130 \approx 2^{77.1}$ 130-round KATAN64 computations. The memory requirement is $2 \cdot 2^{51.05} \approx 2^{52.1}$ state values. The analysis for P_3, P_4 pairs also requires $2^{77.1}$ 130-round KATAN64 computations. P_{right} is $2^{102.1} \cdot 2^{-110} = 2^{-7.9}$. Hence, the complexity for the exhaustive check

Table 6.13: Differential characteristic of KATAN64 E_0 (1 - 56)

Round	$L_2(L_2[0]...L_2[38])$	$L_1(L_1[0]...L_1[24])$	K_a	K_b	Pr.
0	000000000000000000111000000000011100000	0000000000000000000000000	0	0	1
1	000000000000000000000111000000000011100	0000000000000000000000000	0	0	2^{-3}
2	000000000000000000000000111000000000011	0000000000000000000000000	0	0	2^{-4}
3	000000000000000000000000000111000000000	0000000000000000000000000	0	0	2^{-4}
4	000000000000000000000000000000111000000	0000000000000000000000000	0	0	2^{-4}
5	000000000000000000000000000000000111000	0000000000000000000000000	0	0	2^{-6}
6	000000000000000000000000000000000000111	0000000000000000000000000	0	1	2^{-7}
7	000000000000000000000000000000000000000	0000000000000000000000000	0	0	2^{-7}
46	000000000000000000000000000000000000000	0000000000000000000000000	0	1	2^{-7}
47	000000000000000000000000000000000000000	1110000000000000000000000	0	0	2^{-7}
48	000000000000000000000000000000000000000	0001110000000000000000000	0	0	2^{-7}
49	000000000000000000000000000000000000000	0000001110000000000000000	0	0	2^{-7}
50	000000000000000000000000000000000000000	0000000001110000000000000	0	0	2^{-7}
51	000000000000000000000000000000000000000	0000000000001110000000000	0	0	2^{-10}
52	110000000000000000000000000000000000000	0000000000000001110000000	0	0	2^{-10}
53	001110000000000000000000000000000000000	0000000000000000001110000	1	0	2^{-10}
54	111001110000000000000000000000000000000	0000000000000000000001110	0	0	2^{-13}
55	110111001110000000000000000000000000000	0000000000000000000000001	0	0	2^{-14}
56	001110111001110000000000000000000000000	0000000000000000000000000	0	0	2^{-16}

Table 6.14: Differential characteristic of KATAN64 E_1 (57 - 113)

Round	$L_2(L_2[0]...L_2[38])$	$L_1(L_1[0]...L_1[24])$	K_a	K_b	Pr.
56	000000000000000111000000000011100000000	0000000000000000000000000	0	0	1
57	000000000000000000111000000000011100000	0000000000000000000000000	0	0	1
58	000000000000000000000111000000000011100	0000000000000000000000000	0	0	2^{-3}
59	000000000000000000000000111000000000011	0000000000000000000000000	0	0	2^{-4}
60	000000000000000000000000000111000000000	0000000000000000000000000	0	0	2^{-4}
61	000000000000000000000000000000111000000	0000000000000000000000000	0	0	2^{-4}
62	000000000000000000000000000000000111000	0000000000000000000000000	0	0	2^{-6}
63	000000000000000000000000000000000000111	0000000000000000000000000	0	1	2^{-7}
64	000000000000000000000000000000000000000	0000000000000000000000000	0	0	2^{-7}
103	000000000000000000000000000000000000000	0000000000000000000000000	0	1	2^{-7}
104	000000000000000000000000000000000000000	1110000000000000000000000	0	0	2^{-7}
105	000000000000000000000000000000000000000	0001110000000000000000000	0	0	2^{-7}
106	000000000000000000000000000000000000000	0000001110000000000000000	0	0	2^{-7}
107	000000000000000000000000000000000000000	0000000001110000000000000	0	0	2^{-7}
108	000000000000000000000000000000000000000	0000000000001110000000000	0	0	2^{-10}
109	110000000000000000000000000000000000000	0000000000000001110000000	0	0	2^{-10}
110	001110000000000000000000000000000000000	0000000000000000001110000	1	0	2^{-10}
111	111001110000000000000000000000000000000	0000000000000000000001110	0	0	2^{-13}
112	110111001110000000000000000000000000000	0000000000000000000000001	0	0	2^{-14}
113	001110111001110000000000000000000000000	0000000000000000000000000	0	0	2^{-16}

becomes $2^{80} \cdot P_{right} = 2^{72.1}$. In the end, the data complexity is $4 \cdot 2^{51.05} \approx 2^{53.1}$ chosen plaintexts. The computational complexity is $2^{77.1} + 2^{77.1} + 2^{72.1} \approx 2^{78.1}$ 130-round KATAN64 computations. The success probability of our attack is 0.63, while the success probability of the brute force attack with the same complexity is 0.27.

Table 6.15: Partial-matching for KATAN48

#skipped rounds	#bits with unknown diff.			P_{right}
	L_1	L_2	Total	
1	0	0	0	$N^2 \cdot 2^{-96}$
2	1	0	1	$N^2 \cdot 2^{-94}$
3	3	0	3	$N^2 \cdot 2^{-90}$
4	5	1	6	$N^2 \cdot 2^{-84}$
5	7	3	10	$N^2 \cdot 2^{-76}$
6	9	5	14	$N^2 \cdot 2^{-68}$
$r(\geq 4)$	$2r-3$	$2r-7$	$4r-10$	$N^2 \cdot 2^{-116+8r}$

If one subkey bit for the first skipped round is guessed, P_{right} decreases by 2^4.

Table 6.16: Partial-matching for KATAN64

#skipped rounds	#bits with unknown diff.			P_{right}
	L_1	L_2	Total	
1	0	0	0	$N^2 \cdot 2^{-128}$
2	2	1	3	$N^2 \cdot 2^{-122}$
3	5	4	9	$N^2 \cdot 2^{-110}$
4	8	7	15	$N^2 \cdot 2^{-98}$
$r(\geq 3)$	$3r-4$	$3r-5$	$6r-9$	$N^2 \cdot 2^{-146+12r}$

If one subkey bit for the first skipped round is guessed, P_{right} decreases by 2^6.

6.6 Conclusion

In this chapter, we proposed the related-key boomerang attack to 174, 145 and 130 rounds of KATAN32/48/64, respectively, which dramatically improved the number of attacked rounds compared with the previous results. Examples of the right quartet on KATAN32 confirmed the feasibility of our attack. As far as we know, this is the best result achieved on the KATAN family.

Part II

Analysis of Stream Cipher

Chapter 7

Chosen-IV Attack on Py and Pypy

In this chapter, we propose an effective key recovery attack on stream ciphers Py and Pypy with chosen IVs. Our method uses an internal-state correlation based on the vulnerability that the randomization of the internal state in the KSA is inadequate, and it improves two previous attacks proposed by Wu and Preneel (a WP-1 attack and a WP-2 attack). For a 128-bit key and a 128-bit IV, the WP-1 attack can recover a key with 2^{23} chosen IVs and time complexity 2^{72}. First, we improve the WP-1 attack by using the internal-state correlation (called a P-1 attack). For a 128-bit key and a 128-bit IV, the P-1 attack can recover a key with 2^{23} chosen IVs and time complexity 2^{48}, which is $1/2^{24}$ of that of the WP-1 attack. The WP-2 attack is another improvement on the WP-1 attack, and it has been known as the best previous attack against Py and Pypy. For a 128-bit key and a 128-bit IV, the WP-2 attack can recover a key with 2^{23} chosen IVs and time complexity 2^{24}. Second, we improve the WP-2 attack by using the internal-state correlation as well as the P-1 attack (called a P-2 attack). For a 128-bit key and a 128-bit IV, the P-2 attack can recover a key with 2^{23} chosen IVs and time complexity 2^{24}, which is the same capability as that of the WP-2 attack. However, when the IV size is from 64 bits to 120 bits, the P-2 attack is more effective than the WP-2 attack. Thus, the P-2 attack is the known best attack against Py and Pypy.

7.1 Introduction

Symmetric-key cryptography, which uses the same key for encryption and decryption, provides security (e.g. confidentiality and integrity) to data or network. Stream cipher is a class of symmetric-key cryptography, and it is suitable for high-speed applications (e.g. applications for large-scale data processing). A typical stream cipher encrypts a plaintext by XORing it with a pseudo-random sequence (called a keystream) that is generated from a secret key and an initialization vector (IV). The core of the stream cipher is to generate the keystream from the secret key and the IV.

Py [28] is a software-oriented stream cipher, and it was designed by Biham and Seberry in April 2005. Py was submitted to the ECRYPT stream cipher project (eSTREAM) [106], which is a project to identify a portfolio of promising new stream ciphers and takes multi-year effort from 2004 to 2008. The speed of Py is more than 2.5 times faster than that of RC4 on a Pentium Ⅲ processor. RC4 is a widely-used software-oriented stream cipher. Py uses a variable-length key and a variable-length key IV. The key size varies from 1 byte to 256 bytes. The IV size varies from 1 byte to 64 bytes. The recommended parameters for security are that the key size is up to 32 bytes and the IV size is up to 16 bytes. The Py algorithm consists of a key scheduling algorithm (KSA) and a pseudorandom number generation algorithm (PRGA).

Security of a stream cipher is analyzed under the assumption that a part of a keystream is given. Since Biham, who is one of designers of Py, is known as a researcher who did significant works, Py attracted the attention of many cryptanalysts. As the cryptanalysis of Py, a distinguishing attack, which distinguishes a keystream from a random stream, was proposed by Paul, Preneel, and Sekar in December 2005 [107]. Their attack was improved by Crowley in January 2006 [108]. In order to resist these distinguishing attacks, a new version of Py, called Pypy, was proposed by Biham and Seberry in March 2006 [29]. Wu and Preneel pointed out a weakness of the IV setup algorithms of Py and Pypy in August 2006 [30]. The IV setup algorithms of Py and Pypy are identical. The weakness is that two keystreams generated from the chosen IVs can be identical with a high probability. They used the weakness to construct a key recovery attack in September 2006 [109] (called a WP-1 attack). In the WP-1 attack, if the IV size is more than 9 bytes, $(IVsizeb-9)$ bytes of the key can be recovered with $(IVsizeb-4)\times 2^{19}$ chosen IVs, where $IVsizeb$ stand for the size of the IV in bytes. For a 128-bit key and a 128-bit IV, which are recommended parameters for security [28], 7 bytes of the key can be recovered with 2^{23} chosen IVs. Thus, for a 128-bit IV, the WP-1 attack can recover a 128-bit key with 2^{23} chosen IVs and time complexity 2^{72}. Another key recovery attack against Py and Pypy was proposed by Wu and Preneel in May 2007 [31] (called a WP-2 attack). The fundamental idea of the WP-2 attack is same as the WP-1 attack. For a 128-bit key and a 128-bit IV, 13 bytes of the key can be recovered with 2^{23} chosen IVs. Thus, for 128-bit IV, the WP-2 attack can recover a 128-bit key with 2^{23} chosen IVs and time complexity 2^{24}.

In this chapter, we first improve the WP-1 attack [109] (called a P-1 attack). The P-1 attack can recover more 3-byte key information than the WP-1 attack by using the internal-state correlation. The P-1 attack has two new effective processes as compared to those of Wu and Preneel. These two processes are called an expansion process and a comparison process, respectively. In the P-1 attack, if the IV size is more than 7 bytes, $(IVsizeb-6)$ bytes of the key can be recovered with $(IVsizeb-4)\times 2^{19}$ chosen IVs. For a 128-bit key and a 128-bit IV, 10 bytes of the key can be recovered with 2^{23} chosen IVs. Thus, for a 128-bit IV, the P-1 attack can recover a 128-bit key with 2^{23} chosen IVs and time complexity 2^{48}, which is $1/2^{24}$ of that of the WP-1 attack. Second , we improve the WP-2 attack [31] by using the internal-state correlation as well as the P-1 attack (called a P-2 attack). For a 128-bit IV, the P-2 attack can recover a 128-bit key with 2^{23} chosen IVs and complexity 2^{24}, which is the same capability as that of the WP-2 attack. However, when the IV size is from 8 bytes to 15 bytes, the P-2 attack is more effective than the WP-2 attack. The P-2 attack is the known best attack against Py and Pypy.

We explain why we focus on Py and Pypy. We have the following opinion from our experience of stream-cipher analysis: if the internal-state size is much larger than the key size, the KSA needs much computational cost in order to map the key information to the internal state randomly. In other words, if the KSA of a stream cipher is designed to be faster in general to achieve good key agility and the randomization of the large internal state is not sufficient, then bytes of the internal-state tend to correlate each other. We think that attacks based on the internal-state correlation are effective against such stream ciphers, where the internal-state correlation indicates that a part of the internal state includes information on the other internal state. Recall Py and Pypy. Since the internal-state size of Py and Pypy is 1300 bytes and the key size is at most 32 bytes, the internal-state size is much larger than the key size. Indeed, the internal-state size of Py and Pypy is much larger than that of other stream ciphers. Moreover, we think that the computational cost for randomizing the internal state in the KSA of Py and Pypy is not sufficiently-large, although the KSA has good key agility. Hence, we considered that Py and Pypy were especially vulnerable to an attack based on the above opinion.

7.2 Description of Py and Pypy

Py and Pypy are stream ciphers that use a variable-length key and an IV. The key size varies from 1 byte to 256 bytes. The IV size varies from 1 byte to 64 bytes. The recommended parameters for security is that the key size is up to 32 bytes and the IV size is up to 16 bytes. Internal states of Py and Pypy contain two arrays –P and Y– and a 4-byte variable s. P is an array of 256 bytes that contains a permutation of all the values $0, \dots, 255$, and Y is an array of 260 4-byte words indexed as $-3, \dots, 256$. The Py and Pypy algorithms consists of the KSA and PRGA. In the KSA, the key and IV are expanded into internal states of Py and Pypy. In the PRGA, a keystream is generated from internal states of Py and Pypy. Py and Pypy have the same KSAs and different PRGAs. In the each step of the PRGA, Py generates an 8-byte keystream, while Pypy generates only 4-byte keystream to resist the distinguishing attacks [107][108]. The KSA consists of a Key setup, an IV setup1, and an IV setup2. The Key setup initializes Y with the key. The IV setup1 initializes P, s, and EIV with Y and the IV, where EIV is an array with the same size as IV. IV setup2 updates Y, P, and s with EIV. We now provide a description of the Key setup and the IV setup1 and omit the description of the IV setup2 and the PRGA, since our attacks use only the Key setup and the IV setup1.

Our descriptions of the Key setup and the IV Setup are some differences from the original one to explain our attack easily.

7.2.1 Key Setup

In the Key setup, Y is initialized with the key. Figure 1 shows the structure of the Key setup. The main process in the Key setup is divided into three processes listed

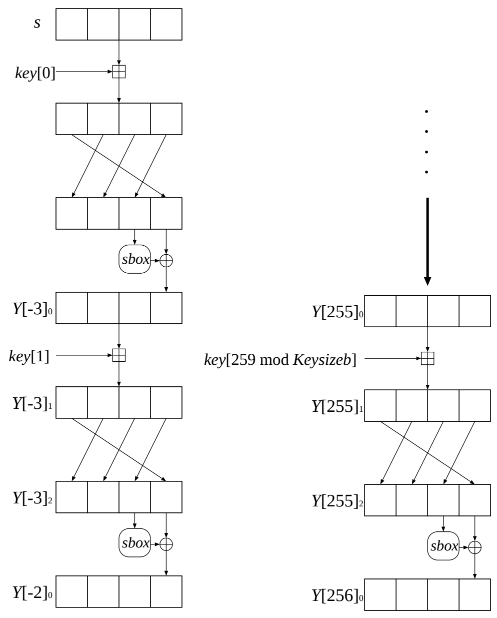


Figure 7.1: Key setup.

as follows:

$$\begin{aligned} Y[i]_1 &= (Y[i]_0 + key[(i+4) \bmod Keysizeb]) \\ &\quad \bmod 2^{32}, && (7.1) \\ Y[i]_{2,j} &= Y[i]_{1,(j-1) \bmod 4} \ (j = 0,1,2,3), && (7.2) \\ Y[i+1]_{0,j} &= \begin{cases} Y[i]_{2,0} \oplus sbox[Y[i]_{2,1}] \ (j=0) \\ Y[i]_{2,j} \ (j = 1,2,3), \end{cases} && (7.3) \end{aligned}$$

where $Y[i]_0, Y[i]_1$, and $Y[i]_2$ are the outputs of each process, $key[i]$ is the value of the $(i+1)$-th byte of the key, $Keysizeb$ is the size of the key in bytes, $Y[i]_{l,j}$ is the value of the $(j+1)$-th byte of $Y[i]_l$, $sbox[\cdot]$ is a substitution box that accepts a 1-byte input and yields a 1-byte output.

First, s is obtained from the key during the preprocessing. Second, the values of $Y[i]_0$ $(-3 \leq i \leq 256)$ are decided from s and Eqs. (7.1)–(7.3), and the values are outputted to IV setup as $Y[i]$ $(-3 \leq i \leq 256)$. In this chapter, we omit the description of the preprocessing stage.

7.2.2 IV Setup1

In IV setup1, P, s, and EIV are initialized with the IV and Y. Figure 2 shows the structure of IV setup1. P can be expressed as

$$\begin{aligned} v &= IV[0] \oplus Y[0]_{0,2}, && (7.4) \\ d &= (IV[1 \bmod IVsizeb] \oplus Y[1]_{0,2})|1, && (7.5) \\ P[i] &= sbox[v + i \cdot d], \ i \in \{0,1,\ldots,255\}, && (7.6) \end{aligned}$$

where v and d are 1-byte variables, $IV[i]$ is the value of the $(i+1)$-th byte of the

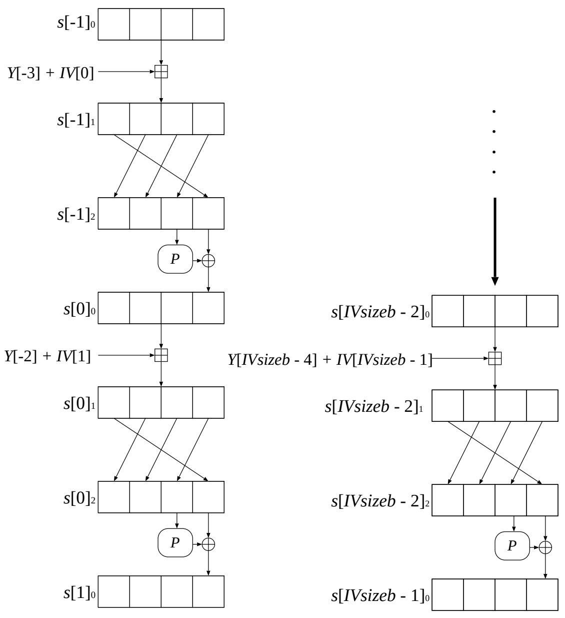


Figure 7.2: IV setup1.

IV, and $IVsizeb$ is the size of the IV in bytes.

The main process in IV setup1 is divided into four processes, which are expressed as

$$\begin{aligned} s[i]_1 &= (s[i]_0 + Y[i-2] + IV[i+1]) \\ &\quad \bmod 2^{32}, \qquad (7.7) \\ s[i]_{2,j} &= s[i]_{1,(j-1) \bmod 4} \; (j=0,1,2,3), \qquad (7.8) \\ EIV[i+1] &= P[s[i]_{2,1}], \qquad (7.9) \\ s[i+1]_{0,j} &= \begin{cases} s[i]_{2,0} \oplus P[s[i]_{2,1}] \; (j=0) \\ s[i]_{2,j} \; (j=1,2,3), \end{cases} \qquad (7.10) \end{aligned}$$

where $s[i]_0, s[i]_1$, and $s[i]_2$ are the outputs of each process.

The values of $EIV[i]$ $(0 \le i \le IVsizeb-1)$ can be determined from the above-mentioned equations. $s[-1]_0$ is given by

$$\begin{aligned} s[-1]_0 = \quad & ((v \ll 24) \oplus (d \ll 16) \oplus (P[254] \ll 8) \\ & \oplus (P[255])) \oplus (Y[-3] + Y[256]). \qquad (7.11) \end{aligned}$$

$EIV[i]$ $(0 \le i \le IVsizeb-1)$ is initialized by using Eqs. (7.7)–(7.10). Next, $EIV[i]$ $(0 \le i \le IVsizeb-1)$ is updated by using

$$\begin{aligned} s'[i]_1 &= (s'[i]_0 + Y[255-i] + IV[i+1]) \\ &\quad \bmod 2^{32}, \qquad (7.12) \\ s'[i]_{2,j} &= s'[i]_{1,(j-1) \bmod 4} \; (j=0,1,2,3), \qquad (7.13) \\ EIV'[i+1] &= EIV[i+1] + P[s'[i]_{2,1}], \qquad (7.14) \\ s'[i+1]_{0,j} &= \begin{cases} s'[i]_{2,0} \oplus P[s'[i]_{2,1}] \; (j=0) \\ s'[i]_{2,j} \; (j=1,2,3), \end{cases} \qquad (7.15) \end{aligned}$$

where s' and EIV' indicate the updated values of s and EIV respectively. $s'[-1]$ is given as $s[IVsizeb-1]_0$.

7.3 WP-1 Attack

We describe the WP-1 attack [109] to understand the P-1 attack in Section 6.4 and the WP-2 attack [31] in Section 6. 5.1. The WP-1 attack is based on the weakness of the IV setup algorithm of Py and Pypy. The weakness is that two keystreams generated from the chosen IVs can be identical with a high probability [30].

7.3.1 Identical Keystreams

From Eqs. (7.4)–(7.6), only 15 bits of the IV ($IV[0]$ and $IV[1]$) are used to initialize the array P. For an IV pair, if the 15 bits are identical, then the resulting arrays P are the same. From [30], if IV_1 and IV_2 have only a two-byte difference and satisfy the following four conditions, then this type of an IV pair results in identical keystreams with probability $2^{-23.2}$.

$$\begin{aligned}
&\text{Condition 1}: && IV_1[i] \oplus IV_2[i] = 1 \quad (1 \le i),\\
&\text{Condition 2}: && IV_1[i+1] \ne IV_2[i+1] \quad (1 \le i),\\
&\text{Condition 3}: && \text{The least significant bit of } IV_1[i] \text{ is } 1 \quad (1 \le i),\\
&\text{Condition 4}: && IV_1[j] = IV_2[j] \quad (0 \le j < i,\ \ i+1 < j \le IVsizeb-1),
\end{aligned}$$

where i is a fixed value.

For the abovementioned IV pair, if two keystreams are identical, then the two $s[i+1]_0$ are the same. This implies

$$\begin{aligned}
&(P[B(s[i-1]_{0,0} + IV_1[i] + Y[-3+i]_{0,0})]\\
&\quad \oplus B(s[i-1]_{0,3} + Y[-3+i]_{0,3} + \xi_{i1}))\\
&\quad +256 + IV_1[i+1]\\
&= (P[B(s[i-1]_{0,0} + IV_2[i] + Y[-3+i]_{0,0}))]\\
&\quad \oplus B(s[i-1]_{0,3} + Y[-3+i]_{0,3} + \xi_{i2}))\\
&\quad +IV_2[i+1],
\end{aligned} \tag{7.16}$$

where $B(x)$ is a function that provides the least significant byte of x, and ξ_{i1} and ξ_{i2} are the carry bits introduced by $IV[i]$ and $Y[-3+i]$. The probability obtaining $\xi_{i1} = \xi_{i2}$ is very close to 1, since $IV[i]$ has a negligible effect on ξ_{i1} and ξ_{i2}.

7.3.2 Recovery of a Part of Y from the Identical IV Pairs

From Eq. (7.16) with the same values of $s[i-1]_{0,0}$ and $s[i-1]_{0,3}$, we can recover values of $B(s[i-1]_{0,0} + Y[-3+i]_{0,0})$ and $B(s[i-1]_{0,3} + Y[-3+i]_{0,3} + \xi_i)$. Equation (7.16) with the same values of $s[i-1]_{0,0}$ and $s[i-1]_{0,3}$ can be generated if the first i bytes of all the IVs are the same. From [109], if there are seven Eqs. (7.16), the values of $B(s[i-1]_{0,0} + Y[-3+i]_{0,0})$ and $B(s[i-1]_{0,3} + Y[-3+i]_{0,3} + \xi_i)$ can be recovered.

After recovering the values of $B(s[i-1]_{0,0} + Y[-3+i]_{0,0})$ and $B(s[i-1]_{0,3} + Y[-3+i]_{0,3} + \xi_i)$ for $i \geq 1$, $s[i]_{0,0}$ can be recovered as follows:

$$\begin{aligned} s[i]_{0,0} &= P[B(s[i-1]_{0,0} + IV[i]^{\theta} + Y[-3+i]_{0,0})] \\ &\quad \oplus B(s[i-1]_{0,3} + Y[-3+i]_{0,3} + \xi_i), \end{aligned} \tag{7.17}$$

where $IV[i]^{\theta}$ is a fixed IV. In the WP-1 attack, we use arbitrary fixed IV^{θ}. When differences are introduced in $IV[i]$ and $IV[i+1]$, all the first i bytes of each IV are chosen to be identical to those of IV^{θ}.

We introduce the IV difference at the $(i+1)$-th and $(i+2)$-th bytes. The first $i+1$ bytes of each IV are identical to those of IV^{θ}. Then, we recover the value of $B(s[i]_{0,0} + Y[-2+i]_{0,0})$. From the value of $B(s[i]_{0,0} + Y[-2+i]_{0,0})$ and $s[i]_{0,0}$, we determine the value of $Y[-2+i]_{0,0}$. Figure 3 shows the method for recovering a part of Y from s.

7.3.3 Generating the Equations

Here, we demonstrate a method for obtaining several equations with the same values of $s[i-1]_{0,0}$ and $s[i-1]_{0,3}$.

To ensure that the same values of $s[i-1]_{0,0}$ and $s[i-1]_{0,3}$ appear in these equations, we need to fix the values of $IV[j]$ $(0 \leq j < i)$. Let the least significant bit of $IV[i]$ and $IV[i+1]$ choose all the 512 values and $IV[j]$ $(0 \leq j < i)$ choose $IV^{\theta}[j]$ $(0 \leq j < i)$. Let $IV[j]$ $(i+2 \leq j \leq IVsize-1)$ choose the optional fixed values. Then we obtain $2^{16}(\approx 255 \times 255)$ desired IV pairs. These 512 IVs are termed as the desired IV group. From [30], this type of IV pair results in identical keystreams with probability $2^{-23.2}$. We obtain $2^{-7.2}(= 2^{-23.2} \times 2^{16})$ identical keystream pairs from one desired IV group. We modify the value of the 7 most significant bits of $IV[i]$ and 3 bits of $IV[i+2]$; we can then obtain $2^{10}(= 2^{7} \times 2^{3})$ desired IV groups. From these desired IV groups, we obtain 7 $(= 2^{10} \times 2^{-7.2})$ Eqs. (7.16). There are $2^{19}(= 2^{7} \times 2^{3} \times 2^{9})$ IVs used in the attack. With $(IVsizeb - 4) \times 2^{19}$ IVs, we can recover $B(s[i-1]_{0,0} + Y[-3+i]_{0,0})$, $B(s[i-1]_{0,3} + Y[-3+i]_{0,3} + \xi_i)$, and $s[i]_{0,0}$ $(2 \leq i \leq IVsizeb-3)$ for IV^{θ}. Thus, we can recover the value of $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb-3)$.

7.3.4 Key Recovery

We now show how to recover the key from $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb-3)$. From Eqs. (7.1)–(7.3), the relation between the key and Y is given by

$$\begin{aligned} &(B(Y[-3+i]_{0,0} + key[i+1] + \xi'_i) \\ &\oplus sbox[B(Y[-3+i+3]_{0,0} + key[i+4])] \\ &= Y[-3+i+4]_{0,0}, \end{aligned} \tag{7.18}$$

where ξ'_i is the carry bit introduced by $key[i+2]$ and $key[i+3]$; it is computed by using $\xi'_i \approx (key[i+2] + Y[-3+i+1]_{0,0}) << 8$. The value of ξ'_i is 0 with a probability approximately 0.5.

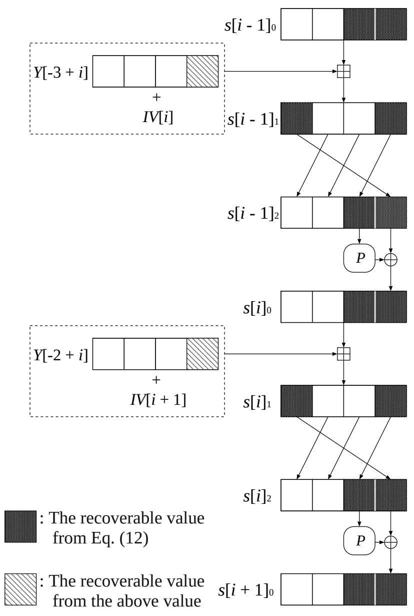


Figure 7.3: Recovery of a part of Y from s.

When $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb - 3)$ are known, Eq. (7.18) becomes an expression that relates $key[i+1]$ and $key[i+4]$ for $(3 \leq i \leq IVsizeb - 7)$. Once we correctly guess the values of $key[4], key[5]$, and $key[6]$, we can determine the other key bytes $key[j]$ $(6 < j \leq IVsizeb - 3)$. Thus, the values of $key[j]$ $(4 \leq j \leq IVsizeb - 3)$ can be restricted to 2^{24} values. This indicates that $(IVsize - 9)$ bytes of the key constitutes the leaked information.

From the description given above, in the WP-1 attack, $(IVsizeb - 9)$ bytes of the key can be recovered with $(IVsizeb - 4) \times 2^{19}$ chosen IVs. Thus, time complexity for recovering a key is $2^{8\times(Keysizeb-(IVsizeb-9))}$. For a 128-bit key and a 128-bit IV, time complexity for recovering a key is 2^{72} $(= 2^{8\times(16-(16-9))})$.

7.4 P-1 Attack

We improve the WP-1 attack with the internal-state correlation. The WP-1 attack and the P-1 attack are different in the process after recovering a part of a Y. The P-1 attack comprises two new effective processes using the internal-state correlation. These two processes are called an expansion process and a comparison process, respectively. In the expansion process, $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb - 3)$ are expanded into the candidate values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb - 3)$. In the comparison process, the values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb - 3)$ are determined by comparing the candidate values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb - 3)$ with $s[i]$ $(2 \leq i \leq IVsizeb - 6)$ recovered from Eqs. (7.16) and (7.17). From these processes, the P-1 attack can use the values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb - 3)$ for recovering the key while the WP-1 attack use only $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb - 3)$. By using the values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb - 3)$, $key[j]$ $(4 \leq j \leq IVsizeb - 3)$ can be recovered. Thus, the P-1 attack can recover more 3-byte key information than

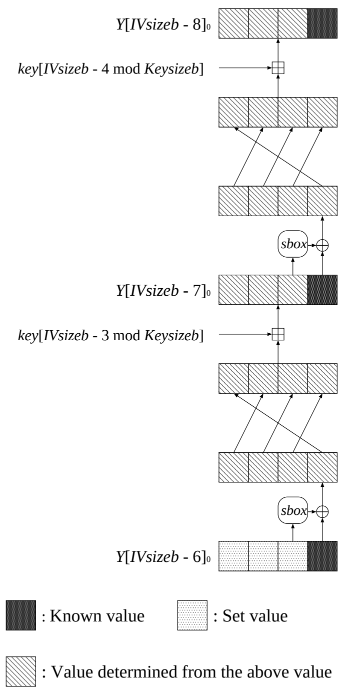

Y[IVsizeb - 8]0
key[IVsizeb - 4 mod Keysizeb]
sbox
Y[IVsizeb - 7]0
key[IVsizeb - 3 mod Keysizeb]
sbox
Y[IVsizeb - 6]0
: Known value
: Set value
: Value determined from the above value

Figure 7.4: Expansion process.

the WP-1 attack.

7.4.1 Expansion Process

Theorem 1. *If* $Y[i]_0$ *and* $Y[i-1]_{0,0}$ *are known, then* $key[(i+3) \bmod Keysizeb]$*,* $Y[i-1]_{0,1}$*,* $Y[i-1]_{0,2}$*, and* $Y[i-1]_{0,3}$ *can be determined from*

$$Y[i-1]_{2,j} = \begin{cases} Y[i]_{0,0} \oplus sbox[Y[i]_{0,1}] \ (j=0) \\ Y[i]_{0,j} \ (j=1,2,3), \end{cases} \tag{7.19}$$

$$Y[i-1]_{1,j} = Y[i-1]_{2,(j+1) \bmod 4} \ (j=0,1,2,3), \tag{7.20}$$

$$key[(i+3) \bmod Keysizeb] = B(Y[i-1]_{1,0} - Y[i-1]_{0,0}), \tag{7.21}$$

$$Y[i-1]_0 = (Y[i-1]_1 - key[(i+3) \bmod Keysizeb]) \bmod 2^{32}. \tag{7.22}$$

Proof. Eqs. (7.19)–(7.22) are derived from Eqs. (7.1)–(7.3). First, $Y[i-1]_1$ is determined from $Y[i]_0$, Eq. (7.19), and Eq. (7.20). Second, $key[(i+3) \bmod Keysizeb]$ is determined from $Y[i-1]_{0,0}$, $Y[i-1]_{1,0}$, and Eq. (7.21). Finally, $Y[i-1]_0$ is determined from $Y[i-1]_1$, $key[(i+3) \bmod Keysizeb]$, and Eq. (7.22). Hence, $Y[i-1]_{0,1}$, $Y[i-1]_{0,2}$, and $Y[i-1]_{0,3}$ can be determined. □

The expansion process is executed after recovering a part of Y from the identical IV pair in the WP-1 attack. In the expansion process, we obtain the candidate values of $Y[-3+i]_{0,1}, Y[-3+i]_{0,2}$, $Y[-3+i]_{0,3}$ $(3 \le i \le IVsizeb - 3)$, and

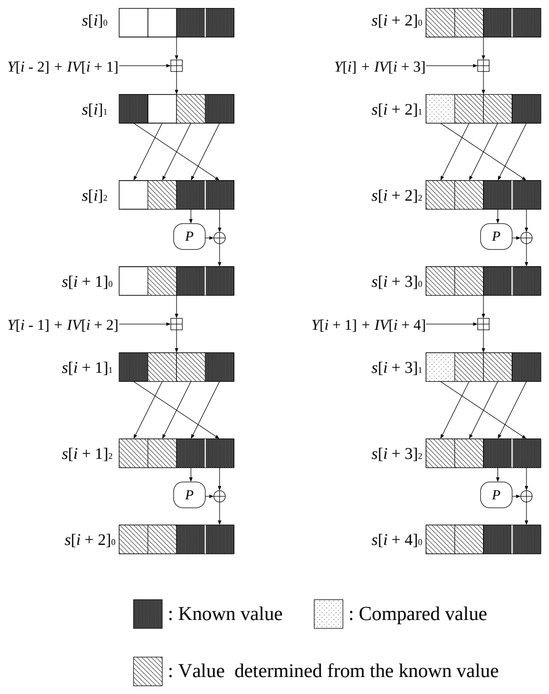


Figure 7.5: Comparison process.

$key[j]$ $(4 \leq j \leq IVsizeb - 3)$ from $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb - 3)$ by going the Key setup algorithm backward. The algorithm of the expansion process is given below. Figure 4 shows the expansion process.

Algorithm 1.

Input: $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb - 3)$.

Output: The candidate values of $Y[-3+i]_{0,1}, Y[-3+i]_{0,2}$, $Y[-3+i]_{0,3}$ $(3 \leq i \leq IVsizeb - 3)$, and $key[j]$ $(4 \leq j \leq IVsizeb - 3)$.

Step 1: If $Y[IVsizeb-6]_{0,1}$, $Y[IVsizeb-6]_{0,2}$, and $Y[IVsizeb-6]_{0,3}$ have been set to all 2^{24} values, terminate this algorithm. Otherwise, set those values that have not been set yet to $Y[IVsizeb-6]_{0,1}$, $Y[IVsizeb-6]_{0,2}$, and $Y[IVsizeb-6]_{0,3}$.

Step 2: Substitute 0 into *num*.

Step 3: Determine $key[((IVsizeb-6)-num+3) \bmod Keysizeb]$, $Y[(IVsizeb-6)-num-1]_{0,1}$, $Y[(IVsizeb-6)-num-1]_{0,2}$, and $Y[(IVsizeb-6)-num-1]_{0,3}$ from Eqs. (7.19)–(7.22), $Y[(IVsizeb-6)-num]_0$, and $Y[(IVsizeb-6)-num-1]_{0,0}$ (Theorem 1).

Step 4: Add 1 to *num*.

Step 5: If $num = IVsizeb - 6$, go to Step 1. Otherwise, go to Step 3.

From the above algorithm, if $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb - 3)$ are known, once we correctly guess the values of $Y[IVsizeb-6]_{0,1}$, $Y[IVsizeb-6]_{0,2}$, and

$Y[IVsizeb-6]_{0,3}$, we can determine the values of $key[j]$ $(4 \leq j \leq IVsizeb-3)$ and $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$. This implies that the values of $key[j]$ $(4 \leq j \leq IVsizeb-3)$ and $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$ can be restricted to only 2^{24} values.

7.4.2 Comparison Process

Theorem 2. *If $s[i]_{0,0}, s[i]_{0,1}, s[i+1]_{0,0}, s[i+1]_{0,1}, s[i+2]_{0,0}, s[i+2]_{0,1}, Y[i-2], Y[i-1], Y[i], IV[i+1], IV[i+2]$, and $IV[i+3]$ are known, $s[i+2]_{1,3}$ can be determined from Eqs. (7.7), (7.8), and (7.10) as follows:*

$$\begin{aligned} s[i]_1 \bmod 2^{16} &= (s[i]_0 \bmod 2^{16} + Y[i-2] \bmod 2^{16} \\ &\quad +IV[i+1]) \bmod 2^{16}, \qquad (7.23) \\ s[i+1]_1 \bmod 2^{24} &= (s[i+1]_0 \bmod 2^{24} \\ &\quad +Y[i-1] \bmod 2^{24} \\ &\quad +IV[i+2]) \bmod 2^{24}. \qquad (7.24) \end{aligned}$$

Proof. Eqs. (7.23) and (7.24) are derived from Eq. (7.7). First, $s[i+1]_{0,2}$ is determined from $s[i]_{0,0}, s[i]_{0,1}$, $Y[i-2]$, $IV[i+1]$, Eq. (7.8), Eq. (7.10), and Eq. (7.23). Second, $s[i+2]_{0,2}$ and $s[i+2]_{0,3}$ are determined from $s[i+1]_{0,0}, s[i+1]_{0,1}$, $s[i+1]_{0,2}$, $Y[i-1]$, $IV[i+2]$, Eq. (7.8), Eq. (7.10), and Eq. (7.24). Finally, $s[i+2]_1$ is determined from $s[i+2]_0$, $Y[i]$, $IV[i+3]$, and Eq. (7.7). Hence, $s[i+2]_{1,3}$ can be determined. □

The comparison process is executed after the expansion process. In the comparison process, we determine the correct values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$ from 2^{24} candidate values and recover the value of $key[j]$ $(4 \leq j \leq IVsizeb-3)$.

For IV^{θ}, $B(s[i-1]_{0,0} + Y[-3+i]_{0,0})$, $B(s[i-1]_{0,3} + Y[-3+i]_{0,3} + \xi_i)$, and $s[i]_{0,0}$ $(2 \leq i \leq IVsizeb-3)$ are known. Hence, $s[i-1]_{1,0}, s[i-1]_{1,3}, s[i]_{0,0}$, and $s[i]_{0,1}$ $(2 \leq i \leq IVsizeb-3)$ can be determined from Eqs. (7.7), (7.8), and (7.10). From Theorem 2, $s[i+2]_{1,3}$ $(2 \leq i \leq IVsizeb-6)$ can be recovered from $s[i]_{0,1}$, $s[i]_{0,0}$ $(2 \leq i \leq IVsizeb-3)$, $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$, and IV^{θ}. Thus, by comparing $s[i+2]_{1,3}$ $(2 \leq i \leq IVsizeb-6)$ recovered from Eqs. (7.16) and Eqs. (7.17) with $s^*[i+2]_{1,3}$ $(2 \leq i \leq IVsizeb-6)$, we can determine the correct values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$, where $s^*[i]$ indicates the value determined from the candidate values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$.

The algorithm of the comparison process is given below. Figure 5 shows the comparison process.

Input: $s[i-1]_{1,0}, s[i-1]_{1,3}, s[i]_{0,1}$, $s[i]_{0,0}$ $(2 \leq i \leq IVsizeb-3)$, the candidate values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$, and IV^{θ}.

Output: $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$.

Step 1: Set one candidate value of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$ to $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$.

Step 2: From Theorem 2, determine $s^*[i+2]_{1,3}$ $(2 \leq i \leq IVsizeb - 6)$ from the candidate values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb - 3)$, $s[i]_{0,1}$, $s[i]_{0,0}$ $(2 \leq i \leq IVsizeb - 3)$, and IV^θ.

Step 3: Compare $s^*[i+2]_{1,3}(2 \leq i \leq IVsizeb-6)$ with $s[i+2]_{1,3}$ $(2 \leq i \leq IVsizeb-6)$ recovered from Eq. (7.16). If $s^*[i+2]_{1,3} = s[i+2]_{1,3}$ $(4 \leq i \leq IVsizeb-3)$, output the candidate values of $Y[-3+i]_0$ $(3 \leq i \leq IVsizeb-3)$ as the correct values. Otherwise, go to Step 1.

We discuss $s^*[i+2]_{1,3}$ $(2 \leq i \leq IVsizeb-6)$ and $s[i+2]_{1,3}$ $(2 \leq i \leq IVsizeb-6)$ recovered from Eq. (7.16) in Step 3. If the values set in Step 1 are correct, then $s^*[i+2]_{1,3} = s[i+2]_{1,3}$ $(2 \leq i \leq IVsizeb-6)$. Theoretically, since the probability $s^*[4]_{1,3} = s[4]_{1,3}$ is 1/256, on comparing $s^*[4]_{1,3}$ with $s[4]_{1,3}$, we can determine 2^{16} candidate values from 2^{24} candidate values. Supposing that the probabilities of $s^*[4]_{1,3} = s[4]_{1,3}$, $s^*[5]_{1,3} = s[5]_{1,3}$, and $s^*[6]_{1,3} = s[6]_{1,3}$ are independent, we can determine 2^8 candidate values by comparing $s^*[4]_{1,3}$ with $s[4]_{1,3}$ and $s^*[5]_{1,3}$ with $s[5]_{1,3}$. We can determine 1 candidate value by comparing $s^*[4]_{1,3}$ with $s[4]_{1,3}$, $s^*[5]_{1,3}$ with $s[5]_{1,3}$, and $s^*[6]_{1,3}$ with $s[6]_{1,3}$.

From the simulation for 10^4 keys, we were able to determine $65536.019 \approx 2^{16}$ candidate values by comparing $s^*[4]_{1,3}$ with $s[4]_{1,3}$, and $255.084 \approx 2^8$ candidate values by comparing $s^*[4]_{1,3}$ with $s[4]_{1,3}$ and $s^*[5]_{1,3}$ with $s[5]_{1,3}$, and 1.384 candidate values by comparing $s^*[4]_{1,3}$ with $s[4]_{1,3}$, $s^*[5]_{1,3}$ with $s[5]_{1,3}$, and $s^*[6]_{1,3}$ with $s[6]_{1,3}$. If the number of comparisons is more than three, only one value can be identified.

From the algorithm given above, we determine the values of $Y[-3+i]_0$ $(3 \leq i \leq IVsize - 3)$ and $key[j](4 \leq j \leq IVsize - 3)$. If $IVsize \geq 10$, these values can be restricted to one value from more than two comparisons. If $IVsize = 9$, these values can be restricted to 2^8 values from two comparisons. If $IVsize = 8$, these values can be restricted to 2^{16} values from one comparison.

7.4.3 Evaluation

In the P-1 attack, $(IVsizeb - 6)$ bytes of the key can be recovered with $(IVsizeb - 4) \times 2^{19}$ chosen IVs. Thus, time complexity for recovering the key is the sum of that of recovering $(IVsizeb-6)$ bytes of the key with $(IVsizeb-4) \times 2^{19}$ chosen IVs and that of exhaustive remaining key search. Since time complexity of exhaustive remaining key search is very larger than that of recovering $(IVsizeb-6)$ bytes of the key with $(IVsizeb-4) \times 2^{19}$ chosen IVs, time complexity for recovering a key is approximated to that of exhaustive remaining key search which is $2^{8\times(Keysizeb-(IVsizeb-6))}$.

Time complexity of the WP-1 attack [109] and that of the P-1 attack for a 128-bit key are shown in Table 1. Time complexity of the WP-1 attack can be calculated from Section 6.3.4 in this chapter. For a 128-bit key and a 128-bit IV, which are recommended parameters for security [28], the P-1 attack can recover the key with 2^{23} chosen IV and time complexity 2^{48} $(= 2^{8\times(16-(16-6))})$. In addition, when $IVsizeb = 8$ and 9, the WP-1 attack is ineffective; in this case, the P-1 attack can recover the key with lower time complexity than the exhaustive key search.

Table 7.1: Time complexity for recovering a 128-bit (16-byte) key.

$IVsizeb$	8	9	10	11	12	13	14	15	16
The WP-1 attack [109]	2^{128}	2^{128}	2^{120}	2^{112}	2^{104}	2^{96}	2^{88}	2^{80}	2^{72}
The P-1 attack	2^{120}	2^{112}	2^{96}	2^{88}	2^{80}	2^{72}	2^{64}	2^{56}	2^{48}
The WP-2 attack [31]	2^{128}	2^{128}	2^{112}	2^{96}	2^{80}	2^{64}	2^{48}	2^{32}	2^{24}
The P-2 attack	2^{112}	2^{96}	2^{64}	2^{48}	2^{32}	2^{24}	2^{24}	2^{24}	2^{24}

7.5 Improvement on the WP-2 attack

In this section, we first explain the WP-2 attack [31] which is the improvement of the WP-1 attack. Second, we propose the P-2 attack which is the improvement on the WP-2 attack. The P-2 attack comprises two new effective processes using the internal-state correlation as well as the P-1 attack. By using these processes, when the IV size is from 8 bytes to 15 bytes, the P-2 attack is more effective than the WP-2 attack.

7.5.1 WP-2 Attack

The WP-2 attack uses the same chosen IVs as the WP-1 attack. From [30], for these chosen IV, if two keystream are identical, we obtain Eq. (7.16) and

$$\begin{aligned}(P[B(s'[i-1]_{0,0}+IV_1[i]+Y[256-i]_{0,0})]\\ \oplus B(s'[i-1]_{0,3}+Y[256-i]_{0,3}+\xi_{i1}))\\ +256+IV_1[i+1]\\ =(P[B(s'[i-1]_{0,0}+IV_2[i]+Y[256-i]_{0,0}))]\\ \oplus B(s'[i-1]_{0,3}+Y[256-i]_{0,3}+\xi_{i2}))\\ +IV_2[i+1].\end{aligned} \tag{7.25}$$

The WP-2 attack uses these two equations while the WP-1 attack uses only Eq. (7.16). By applying an attack similar to the WP-1 attack, we can recover $B(s'[i-1]_{0,0}+Y[256-i]_{0,0})$, $B(s'[i-1]_{0,3}+Y[256-i]_{0,3}+\xi_i)$, and $s'[i]_{0,0}$ $(2 \leq i \leq IVsizeb-3)$ as well as $B(s[i-1]_{0,0}+Y[-3+i]_{0,0})$, $B(s[i-1]_{0,3}+Y[-3+i]_{0,3}+\xi_i)$, and $s[i]_{0,0}$ $(2 \leq i \leq IVsizeb-3)$ for IV^θ. Thus, we can recover the value of $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb-3)$ and $Y[256-i]_{0,0}$ $(3 \leq i \leq IVsizeb-3)$.

When $Y[-3+i]_{0,0}$ $(3 \leq i \leq IVsizeb-3)$ and $Y[256-i]_{0,0}$ $(3 \leq i \leq IVsizeb-3)$ are known, Eq. (7.18) becomes the expressions that relates $key[j+1]$ and $key[j+4]$ $(3 \leq j \leq IVsizeb-7,\ 262-ivsizeb \leq j \leq 252)$. Thus, there are $2 \times (IVsizeb-9)$ expressions (7.18) linking the key bytes. For a 128-bit key and a 128-bit IV, 14 expressions (7.18) can be obtained : 7 expressions (7.18) linking $key[j]$ and $key[j+3]$ $(4 \leq j \leq 10)$, and another 7 expressions (7.18) linking $key[j]$ and $key[j+3 \bmod 16]$ $(7 \leq j \leq 13)$. There are 13 bytes involved in these 14 expressions (7.18). Since these 14 expressions are sufficient to recover the involved 13 key bytes, time complexity for recovering a key is 2^{24}. For $IVsizeb \leq 15$, since the number of expressions

(7.18) is less than that of involved key bytes, time complexity for recovering a key is $2^{8\times(Keysizeb-2\times(IVsizeb-9))}$. For a 128-bit key and a 120-bit IV, time complexity for recovering a key is 2^{32} ($= 2^{8\times(16-2\times(15-9))}$). The number of the chosen IVs used for the attack is $(IVsizeb - 4) \times 2^{19}$ as well as the WP-1 attack.

7.5.2 P-2 Attack

The P-2 attack has the expansion process and the comparison process after recovering a part of a Y in the WP-2 attack as well as the P-1 attack.

In the expansion process, we obtain 2^{24} candidate values of $Y[-3+i]_{0,1}, Y[-3+i]_{0,2}$, $Y[-3+i]_{0,3}$ $(3 \le i \le IVsizeb - 3)$, and $key[j]$ $(4 \le j \le IVsizeb - 3)$ from $Y[-3+i]_{0,0}$ $(3 \le i \le IVsizeb - 3)$ by using Algorithm 1. Moreover, we can obtain 2^{24} candidate values of $Y[256-i]_{0,1}, Y[256-i]_{0,2}, Y[256-i]_{0,3}$ $(3 \le i \le IVsizeb - 3)$, and $key[j \bmod 16]$ $(23 - IVsizeb \le j \le 16)$ from $Y[256-i]_{0,0}$ $(3 \le i \le IVsizeb - 3)$ by using the algorithm similar to Algorithm 1. Thus, the values of $key[j]$ $(4 \le j \le IVsizeb - 3)$ and $Y[-3+i]_0$ $(3 \le i \le IVsizeb - 3)$, and $key[j \bmod 16]$ $(23 - IVsizeb \le j \le 16)$ and $Y[256-i]_0$ $(3 \le i \le IVsizeb - 3)$ can be restricted to only 2^{24} values respectively.

In the comparison process, we determine the values of $key[j]$ $(4 \le j \le IVsizeb - 3)$ and $key[j \bmod 16]$ $(23 - IVsizeb \le j \le 16)$. From the P-1 attack, we can determine the correct value of $Y[-3+i]_0$ $(3 \le i \le IVsizeb - 3)$ and recover the value of $key[j]$ $(4 \le j \le IVsizeb - 3)$. For IV^θ, $B(s'[i-1]_{0,0} + Y[256-i]_{0,0})$, $B(s'[i-1]_{0,3} + Y[256-i]_{0,3} + \xi_i)$, and $s'[i]_{0,0}$ $(2 \le i \le IVsizeb - 3)$ are known from the WP-2 attack. Hence, $s'[i-1]_{1,0}, s'[i-1]_{1,3}, s'[i]_{0,0}$, and $s'[i]_{0,1}$ $(2 \le i \le IVsizeb - 3)$ can be determined from Eqs. (7.12), (7.13), and (7.15). $s'[i+2]_{1,3}$ $(2 \le i \le IVsizeb - 6)$ can be recovered from $s'[i]_{0,1}$, $s'[i]_{0,0}$ $(2 \le i \le IVsizeb - 3)$, $Y[256-i]_0$ $(3 \le i \le IVsizeb - 3)$, and IV^θ. By using the algorithm similar to Algorithm 2, we can determine the correct value of $Y[256-i]_0$ $(3 \le i \le IVsizeb - 3)$ and recover the value of $key[j \bmod 16]$ $(23 - IVsizeb \le j \le 16)$.

From the algorithm given above, we determine the values of $key[j] (4 \le j \le IVsize - 3)$ and $key[j \bmod 16]$ $(23 - IVsizeb \le j \le 16)$. If $IVsize \ge 10$, these values can be restricted to one value from more than two comparisons. If $IVsize = 9$, these values can be restricted to 2^8 values from two comparisons. If $IVsize = 8$, these values can be restricted to 2^{16} values from one comparison.

7.5.3 Evaluation

In the P-2 attack, we can recover the values of $key[j] (4 \le j \le IVsize - 3)$ and $key[j \bmod 16]$ $(23 - IVsizeb \le j \le 16)$ with $(IVsizeb - 4) \times 2^{19}$ chosen IVs. For a 128-bit key and a 128-bit IV, $key[j \bmod 16]$ $(4 \le j \le 16)$ can be recovered with 2^{23} chosen IVs. Since remaining key values are $key[1]$, $key[2]$, and $key[3]$, time complexity for recovering a key is 2^{24}. Similarly, for a 128-bit key and an 88-bit IV, since remaining key values are $key[1]$, $key[2]$, $key[3]$, $key[9]$, $key[10]$, and $key[11]$, time complexity for recovering a key is 2^{48}.

Time complexity of the P-2 attack and that of the WP-2 attack [31] for a 128-bit key are shown in Table 1. Time complexity of the WP-2 attack can be calculated from Sect .6.5.1 in this chapter. When IV size is from 8 to 15 byte, the P-2 attack is more effective than the WP-2 attack. In addition, when $IVsizeb = 8$, 9, the WP-2 attack is ineffective; in this case, the P-2 attack can recover the key with lower time complexity than the exhaustive key search and is more effective than the P-1 attack.

7.6 Conclusion

In this chapter, we have proposed the P-1 attack and P-2 attack which are improvements on the WP-1 attack and WP-2 attack, respectively. These proposed attacks comprise two new effective processes using the internal-state correlation. As a result, the P-1 attack can recover more 3-byte key information than the WP-1 attack. For a 128-bit key and a 128-bit IV, the P-1 attack can recover the key with time complexity of 2^{48}, which is $1/2^{24}$ of that of the WP-1 attack. When $IVsizeb$ is from 8 to 15 bytes, The P-2 attack can recover the 128-bit key with less time complexity than the WP-2 attack. In particular, when $IVsizeb$ is from 13 to 16 byte, the P-2 attack can recover a 128-bit key with time complexity 2^{24}. In Py and Pypy, since a variable-length IV is used, in some cases the IV may range in size from 8 bytes to 15 bytes. Thus, the P-2 attack is the known best attack against Py and Pypy.

Our attacks show that the attack based on the internal-state correlation is effective against a stream cipher such that the internal-state size is large and the randomization of the internal state in the KSA is not sufficient. Thus, the ideas of our attacks are applicable to Py-like ciphers.

Chapter 8

Slide Cryptanalysis of RAKAPOSHI

This chapter gives a first security evaluation of a lightweight stream cipher RAKAPOSHI. In particular, we analyze a slide property of RAKAPOSHI such that two different Key-IV pairs generate same keystream but n-bit shifted. To begin with, we demonstrate that any Key-IV pair has a corresponding *slide* Key-IV pair that generates an n-bit shifted keystream with probability of 2^{-2n}. In order to experimentally support our results, some examples of such pairs are given. Then, we show that this property is able to be converted into key recovery attacks on RAKAPOSHI. In the related-key setting, our attack based on the slide property can recover a 128-bit key with time complexity of 2^{41} and 2^{38} chosen IVs. Moreover, by using a variant of slide property called partial slide pair, this attack is further improved, and then a 128-bit key can be recovered with time complexity of 2^{33} and 2^{30} chosen IVs. Finally, we present a method for speeding up the brute force attack by a factor of 2 in the single key setting.

8.1 Introduction

With the recent large deployment of low resource devices such as RFID tags and sensor nodes, the demand for security in resource-constrained environments has been dramatically increased. As a result, design and analysis of lightweight ciphers has received a lot of attention from the cryptographic community. A number of lightweight primitives are proposed, e.g., block ciphers : PRESENT [46], KATAN [24], LED [20] and Piccolo [21], and hash functions : Quark [110], PHOTON [111] and SPONGENT [112]. As for lightweight stream ciphers, Grain v1 [113], Trivium [114] and MICKY2.0 [115] are selected by eSTREAM project for hardware applications with highly restricted resources [106]. In spite of considerable efforts in a multi-years project, it is still debatable that design and analysis of stream ciphers are mature enough. Indeed, after the end of this project in 2008, F-FCSR-H [116], which is initially contained in the final portfolio, and the 128-bit version of Grain are broken [117, 118].

Cid, Kiyomoto, and Kurihara proposed a lightweight stream cipher RAKAPOSHI [32] after the eSTREAM project. RAKAPOSHI is a hardware-oriented stream cipher accepting a 128-bit key and a 192-bit IV, and employs a bit-oriented

Dynamic Linear Feedback Shift Register. This structure is also adopted in K2 v2.0 [119], which is recently selected in ISO standard stream ciphers [120]. RAKAPOSHI is considered as a variant of the K2 v2.0 for the low-cost hardware implementation. Its performance properties in hardware are comparable to stream ciphers selected in eSTREAM, e.g., the circuit size of RAKAPOSHI is estimated as about 3K gates. In addition, RAKAPOSHI can provide a 128 bit security while Grain and Trivium have only an 80-bit security. Thus, designers claim that RAKAPOSHI can complement the eSTREAM portfolio, and increase the choice of secure lightweight stream ciphers. Despite its notable features of design and implementations, there exist only designers' self evaluations, i.e., no external cryptanalysis has been published so far.

In this chapter, we give a first security evaluation of the lightweight stream cipher RAKAPOSHI. In particular, we deeply analyze a slide property of RAKAPOSHI such that two different Key-IV pairs generate same keystream but n-bit shifted. This property mainly exploits a weakness of an initialization algorithm, and has been applied to Grain v1 stream cipher [121, 122]. To begin with, by exploiting the self-similarity of the initialization algorithm of RAKAPOSHI, we show that any Key-IV pair has a corresponding *slide* Key-IV pair that generates an n-bit shifted keystream with probability of 2^{-2n}. For $n = 1$, a Key-IV pair has a corresponding Key-IV pair that generates a only 1-bit shifted keystream with probability of 2^{-2}. Besides, we introduce a variant of the slide property, which is called *partial slide property*, that occurs with higher probability than the basic slide property. Then we utilize these properties in order to construct related-key attacks on RAKAPOSHI. Our attack using the slide property can recover a 128-bit key with time complexity of 2^{41} and 2^{38} chosen IVs. Moreover, by using the partial slide property, this attack is further improved, and then a 128-bit key can be recovered with time complexity of 2^{33} and 2^{30} chosen IVs. This result reveals that RAKAPOSHI is practically vulnerable to the related-key attack based on the slide property. Finally, using this variant of the slide property, we give a method for speeding up the brute force attack in the single key setting by a factor of 2.

8.2 Description of RAKAPOSHI

RAKAPOSHI is a stream cipher supporting a 128-bit key and a 192-bit IV. At time t, RAKAPOSHI consists of a 128-bit Non-linear Feedback Shift Register (NFSR) : $A^t = \{a_t, a_{t+1}, \ldots, a_{t+127}\}$ ($a_t \in \{0, 1\}$), a 192-bit Linear Feedback Shift Register (LFSR) : $B^t = \{b_t, b_{t+1}, \ldots, b_{t+191}\}$ ($b_t \in \{0, 1\}$) and an 8-to-1 nonlinear function v (see Fig. 1). Since RAKAPOSHI employs the bit-oriented Dynamic Linear Feedback Shift Register (DLFSR), two bits of the register A are used for dynamically updating the feedback function of the register B.

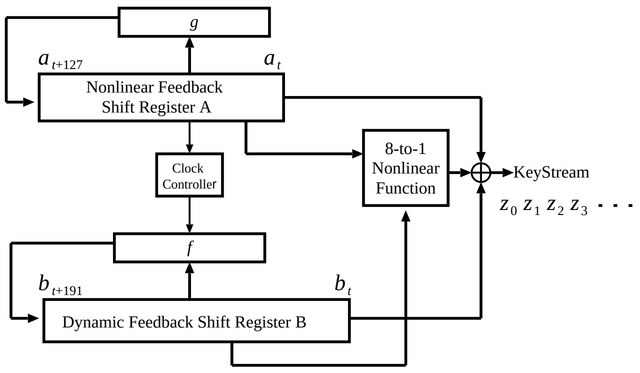

g
a_{t+127}
a_t
Nonlinear Feedback Shift Register A
Clock Controller
8-to-1 Nonlinear Function
KeyStream
$z_0\, z_1\, z_2\, z_3 \ldots$
f
b_{t+191}
b_t
Dynamic Feedback Shift Register B

Figure 8.1: RAKAPOSHI stream cipher

The NFSR A^t and the LFSR B^t are updated as follows:

$$\begin{aligned}
a_{t+128} &= g(a_t, a_{t+6}, a_{t+7}, a_{t+11}, a_{t+16}, a_{t+28}, a_{t+36}, a_{t+45}, a_{t+55}, a_{t+62}) \\
&= 1 \oplus a_t \oplus a_{t+6} \oplus a_{t+7} \oplus a_{t+11} \oplus a_{t+16} \oplus a_{t+28} \oplus a_{t+36} \\
&\quad \oplus a_{t+45} \oplus a_{t+55} \oplus a_{t+62} \oplus a_{t+7}a_{t+45} \oplus a_{t+11}a_{t+55} \\
&\quad \oplus a_{t+7}a_{t+28} \oplus a_{t+28}a_{t+55} \oplus a_{t+6}a_{t+45}a_{t+62} \oplus a_{t+6}a_{t+11}a_{t+62}, \\
b_{t+192} &= f(b_t, b_{t+14}, b_{t+37}, b_{t+41}, b_{t+49}, b_{t+51}, b_{t+93}, b_{t+107}, b_{t+120}, \\
&\quad b_{t+134}, b_{t+136}, b_{t+155}, b_{t+158}, b_{t+176}, c_0, c_1) \\
&= b_t \oplus b_{t+14} \oplus b_{t+37} \oplus b_{t+41} \oplus b_{t+49} \oplus b_{t+51} \oplus b_{t+93} \\
&\quad \oplus \overline{c_0} \cdot \overline{c_1} \cdot b_{t+107} \oplus \overline{c_0} \cdot c_1 \cdot b_{t+120} \oplus c_0 \cdot \overline{c_1} \cdot b_{t+134} \\
&\quad \oplus c_0 \cdot c_1 \cdot b_{t+136} \oplus \overline{c_0} \cdot b_{t+155} \oplus c_0 \cdot b_{t+158} \oplus b_{t+176},
\end{aligned}$$

where $\oplus$ is a bit-wise XOR, $\overline{x}$ is complement of x, and c_0 and c_1 are a_{t+41} and a_{t+89}, respectively. The 8-to-1 nonlinear function v is expressed as

$$s_t = v(a_{t+67}, a_{t+127}, b_{t+23}, b_{t+53}, b_{t+77}, b_{t+81}, b_{t+103}, b_{t+128}).$$

The non-linear function v is given as follows.

$$\begin{aligned}
&v(x_0,x_1,x_2,x_3,x_4,x_5,x_6,x_7) = \\
&x_0x_1x_2x_3x_4x_5x_6 + x_0x_1x_2x_3x_4x_5 + x_0x_1x_2x_3x_4x_6 + \\
&x_0x_1x_2x_3x_5x_6x_7 + x_0x_1x_2x_3x_5x_6 + x_0x_1x_2x_3x_5x_7 + \\
&x_0x_1x_2x_3x_5 + x_0x_1x_2x_3x_6x_7 + x_0x_1x_2x_4x_5x_6 + \\
&x_0x_1x_2x_4 + x_0x_1x_2x_5x_6 + x_0x_1x_2x_5x_7 + x_0x_1x_2x_7 + \\
&x_0x_1x_2 + x_0x_1x_3x_4x_5x_6x_7 + x_0x_1x_3x_4x_5x_7 + x_0x_1x_3x_4x_5 + \\
&x_0x_1x_3x_4x_7 + x_0x_1x_3x_4 + x_0x_1x_3x_6 + x_0x_1x_4x_5x_6x_7 + \\
&x_0x_1x_4x_5x_6 + x_0x_1x_4x_5x_7 + x_0x_1x_4x_6x_7 + x_0x_1x_4x_7 + \\
&x_0x_1x_5x_6x_7 + x_0x_1x_5x_6 + x_0x_1x_5 + x_0x_1x_6 + x_0x_1 + \\
&x_0x_2x_3x_4x_5x_6 + x_0x_2x_3x_4x_5x_7 + x_0x_2x_3x_4 + x_0x_2x_3x_5x_6x_7 + \\
&x_0x_2x_3x_5x_6 + x_0x_2x_3x_5x_7 + x_0x_2x_3x_6 + x_0x_2x_4x_5x_6x_7 + \\
&x_0x_2x_5x_6 + x_0x_2x_5 + x_0x_2x_6x_7 + x_0x_2x_7 + x_0x_3x_4x_5x_6x_7 + \\
&x_0x_3x_4x_5x_6 + x_0x_3x_4x_5x_7 + x_0x_3x_4x_5 + x_0x_3x_4x_7 + \\
&x_0x_3x_5x_6x_7 + x_0x_3x_5 + x_0x_3x_6 + x_0x_3 + x_0x_4x_5x_6 + \\
&x_0x_4x_6x_7 + x_0x_5x_6 + x_0x_6 + x_0 + x_1x_2x_3x_4 + x_1x_2x_3x_5x_6 + \\
&x_1x_2x_3x_5x_7 + x_1x_2x_3x_5 + x_1x_2x_3 + x_1x_2x_4x_5x_6 + x_1x_2x_4x_6 + \\
&x_1x_2x_4 + x_1x_2x_5 + x_1x_2 + x_1x_3x_4x_5x_6x_7 + x_1x_3x_4x_5x_7 + \\
&x_1x_3x_4x_6x_7 + x_1x_3x_4x_6 + x_1x_3x_4 + x_1x_3x_5x_6 + x_1x_3x_5 + \\
&x_1x_3x_6 + x_1x_3x_7 + x_1x_4x_5x_6x_7 + x_1x_4x_5x_7 + x_1x_5x_6 + \\
&x_1x_5x_7 + x_1x_5 + x_1x_6x_7 + x_1x_6 + x_1 + x_2x_3x_4x_5x_6 + \\
&x_2x_3x_4x_5x_7 + x_2x_3x_4x_5 + x_2x_3x_4x_6x_7 + x_2x_3x_4 + x_2x_3x_5x_7 + \\
&x_2x_3x_6x_7 + x_2x_3x_6 + x_2x_4x_5x_6 + x_2x_4x_5x_7 + x_2x_4x_5 + \\
&x_2x_4x_6x_7 + x_2x_4x_6 + x_2x_4x_7 + x_2x_4 + x_2x_5x_6x_7 + \\
&x_2x_6x_7 + x_2x_6 + x_2x_7 + x_3x_4x_5x_6x_7 + x_3x_4x_5 + x_3x_4x_6x_7 + \\
&x_3x_4x_6 + x_3x_4x_7 + x_3x_5x_6x_7 + x_3x_6x_7 + x_3x_6 + x_3x_7 + \\
&x_4x_5x_6 + x_4x_5 + x_5x_6x_7 + x_5x_6 + x_5 + x_6 + x_7.
\end{aligned}$$

8.2.1 Initialization Process

A 128-bit key $K = \{k_0, k_1, \ldots, k_{127}\}$ ($k_i \in \{0,1\}$) and an initialization vector $IV = \{iv_0, iv_1, \ldots, iv_{191}\}$ ($iv_i \in \{0,1\}$) are loaded into the register A and B as follows:

$$a_i = k_i \ (0 \le i \le 127), \quad b_i = iv_i \ (0 \le i \le 191).$$

The initialization process updates the state 448 times without the keystream generation. It consists of a stage 1 (320 cycles) and a stage 2 (128 cycles). In the stage 1, the output of the nonlinear function s_t is fed back to register B^t, i.e., s_t is XORed with b_{t+192}. In the stage 2, the output of the nonlinear function s_t is fed back to register A^t, i.e., s_t is XORed with a_{t+128}. The usage of s_t is only the difference between the stage 1 and the stage 2.

After the initialization process, the state $S^{448} = (\{A^{448}, B^{448}\})$ is obtained.

8.2.2 Keystream Generation

For $t \geq 448$, an internal state $S^t = (A^t, B^t)$ generates a keystream bit z_{t-448} such that $z_{t-448} = b_t \oplus a_t \oplus s_t$ with updating the internal state. Note that the fixed key and IV pair must be changed after 2^{64} keystream bits are generated.

8.3 Slide Property of Stream Cipher

If two different keys always convert same plaintexts into same ciphertexts, such a key pair is called equivalent key in terms of that these keys are functionally equivalent. Since stream ciphers additionally use IV for generating a keystream, equivalent Key-IV pairs can also be defined. Here, a ciphertext is obtained by XORing a plaintext with a keystream in stream ciphers. Thus, an equivalent Key-IV pair essentially means the pair generating same keystreams. The existence of these pairs indicates that effective (K, IV) space is smaller than the expected value which is the sum of the K and IV size.

In stream ciphers, a variant of equivalent (K, IV) called *slide Key-IV pairs* is also defined in [121, 122]. A slide Key-IV pair generates same keystream but $w \cdot n$-bit shifted, where w is the size of the word z_t in the keystream, e.g., RAKAPOSHI is $w = 1$. Though the existence of this pair does not directly affect the effective (K, IV) space unlike the case of equivalent Key-IV pairs, it has the following interesting property.

Let $(K'_{(n)}, IV'_{(n)})$ be $w \cdot n$-bit slide Key-IV pair of (K, IV). In other words, $(K'_{(n)}, IV'_{(n)})$ generates the $w \cdot n$-bit shifted keystream with respect to that of (K, IV) such that $z'_t = z_{t+n}$ for $0 < t$. Suppose that plaintexts $P = \{p_0, p_1, \dots, p_L\}$ and $P' = \{p'_0, p'_1, \dots, p'_L\}$ are encrypted with (K, IV) and $(K'_{(n)}, IV'_{(n)})$, respectively. Then, ciphertexts $C = \{c_0, c_1, \dots, c_L\}$ and $C' = \{c'_0, c'_1, \dots, c'_L\}$ are as follows:

$$\begin{aligned}
C = \{c_0, c_1, \dots, c_L\} &= \{p_0 \oplus z_0, p_1 \oplus z_1, \dots, p_L \oplus z_L\}, \\
C' = \{c'_0, c'_1, \dots, c'_L\} &= \{p'_0 \oplus z'_0, p'_1 \oplus z'_1, \dots, p'_L \oplus z'_L\} \\
&= \{p'_0 \oplus z_n, p'_1 \oplus z_{n+1}, \dots, p'_L \oplus z_{n+L}\}.
\end{aligned}$$

If an attacker can gets above ciphertexts which are generated from $w \cdot n$-bit slide Key-IV pairs, he can obtain information of plaintexts from only ciphertexts without knowledge of keys by XORing $w \cdot n$-bit shifted C to C' as follows:

$$\begin{aligned}
c_{n+t} \oplus c'_t &= p_{n+t} \oplus z_{n+t} \oplus p'_t \oplus z_{n+t} \\
&= p_{n+t} \oplus p'_t.
\end{aligned}$$

At first glance, this assumption seems to be very strong. However, it corresponds to the related-key and chosen IV setting for some classes of stream ciphers[1]. Beside, a slide Key-IV pair can be used not only for exposing plaintext information but also for related-key key recovery attacks [121, 122]. Moreover, it may be able to utilize for speeding up the key search in the single key setting [122].

Therefore, the slide property is a very useful tool of analyses and evaluations of the security of stream ciphers.

[1]It highly depends on structures and algorithms of target stream ciphers.

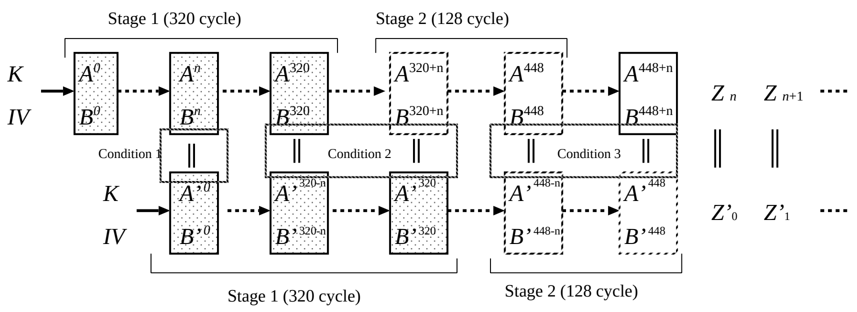


Figure 8.2: n-bit slide pair of RAKAPOSHI

8.4 Slide Property of RAKAPOSHI

In this section, we analyze a slide property of RAKAPHOSHI stream cipher.

8.4.1 Conditions of Slide Pairs

For RAKAPOSHI, a (K, IV) pair has a corresponding n-bit *slide* pair $(K'_{(n)}, IV'_{(n)})$ that generates an n-bit shifted keystream for $0 \le n < 320$, if these pairs satisfy following conditions:

Condition 1 : $S^n(=\{A^n, B^n\}) = S'^0(=\{A'^0, B'^0\}),$

Condition 2 : $s^{320+i} = 0 \quad (0 \le i < n),$

Condition 3 : $s^{448+i} = 0 \quad (0 \le i < n),$

where S'^t is a state generated from $(K'_{(n)}, IV'_{(n)})$ at time t. Figure 2 illustrates these conditions for an n-bit slide pair.

Assume that the condition 1 holds, $S^{320}(=\{A^{320}, B^{320}\})$ and $S'^{320-n}(=\{A'^{320-n}, B'^{320-n}\})$ are identical, because S^n and S'^0 are updated in the same manner during the stage 1.

However, S^{320} and S'^{320-n} are updated by different update processes in the stage 1 and 2, respectively. As mentioned before, the difference of these stages is only usage of s_t, i.e., s_t is XORed with b_{t+192} in the stage 1 while it is XORed with a_{t+128} in the stage 2. When the condition 2 holds, the relation of $s^{320+i} = s'^{320-n+i} = 0$ is obtained for $0 \le i < n$. It allows us to omit these differences of the stage 1 and 2, and then $S^{320+n}(=\{A^{320+n}, B^{320+n}\}) = S'^{320}(=\{A'^{320}, B'^{320}\})$. After that, since these states are updated in the same manner during the stage 2, $S^{448}(=\{A^{448}, B^{448}\})$ and $S'^{448-n}(=\{A'^{448-n}, B'^{448-n}\})$ are surely identical.

In the keystream generation, s_t is used for generating a keystream bits, and does not affect the state updating. Therefore, the condition 3 ensures that $S^{448+n}(=\{A^{448+n}, B^{448+n}\})$ and $S'^{448}(=\{A'^{448}, B'^{448}\})$ are identical. It means that (K', IV') produces n-bit sliding keystream with respect to (K, IV). In other words, the following equations holds, $z_i = z'_{i-n}$ $(n \le i < 2^{64})$.

Table 8.1: Experimental results of probability that a Key-IV pair have a n bit slide pair.

n	Theoretical value	Experimental value
1	2^{-2}	$2^{-2.0011}$ (4191041/16777216)
2	2^{-4}	$2^{-4.0039}$ (1045695/16777216)
3	2^{-6}	$2^{-6.0027}$ (261636/16777216)
4	2^{-8}	$2^{-7.9942}$ (65797/16777216)
5	2^{-10}	$2^{-9.9951}$ (16439/16777216)
6	2^{-12}	$2^{-12.0084}$ (4072/16777216
7	2^{-14}	$2^{-14.0084}$ (1018/16777216)
8	2^{-16}	$2^{-16.0284}$ (251/16777216)
9	2^{-18}	$2^{-17.933}$ (67/16777216)
10	2^{-20}	$2^{-19.830}$ (18/16777216)

8.4.2 Evaluation

Let us estimate how many n -bit slide pairs exist in RAKAPOSHI stream cipher. The condition 1 is expressed as

$$\begin{aligned} A^n &= \{k_n, k_{n+1}, \ldots, k_{127}, x_0, \ldots, x_{n-1}\} \\ &= \{k'_0, k'_1, \ldots, k'_{127}\} = A'^0, \\ B^n &= \{iv_n, iv_{n+1}, \ldots, iv_{191}, y_0, \ldots, y_{n-1}\} \\ &= \{iv'_0, iv'_1, \ldots, iv'_{191}\} = B'^0, \end{aligned}$$

where $x_t = g(a_t, a_{t+6}, a_{t+7}, a_{t+11}, a_{t+16}, a_{t+28}, a_{t+36}, a_{t+46}, a_{t+55}, a_{t+62})$ and $y_t = f(b_t, b_{t+14}, b_{t+37}, b_{t+41}, b_{t+49}, b_{t+51}, b_{t+93}, b_{t+107}, b_{t+120}, b_{t+134}, b_{t+136}, b_{t+155}, b_{t+158}, b_{t+176}, a_{t+41}, a_{t+89}) \oplus s_t$. Since the state size and the sum of K and IV size are same, a (K, IV) pair surely has one pair of (K'_n, IV'_n) satisfying the condition 1 regardless of the value of n.

On the other hand, conditions 2 and 3 depend on the value of n. The probability that a (K, IV) pair satisfies the conditions 2 and 3 is $2^{-2n}(= 2^{-n} \times 2^{-n})$. Therefore, any (K, IV) pair theoretically has an n-bit slide pair $(K'_{(n)}, IV'_{(n)})$ that generates an n-bit shifted keystream with probability of 2^{-2n}. We have confirmed the correctness of this theoretical values by testing 2^{24} random chosen (K, IV) pairs for $n = 0, \ldots, 10$. Table 1 shows experimental results of probability that a Key-IV pair have an n bit slide pair for 2^{24} randomly-chosen Key-IV pairs. It is confirmed that our theoretical values are correctly approximated. Table 2 gives examples of 1, 5 and 10 bits slide pairs. In addition, we can say that a (K, IV) pair having $(K'_{(n)}, IV'_{(n)})$ pairs also has $(K'_{(1)}, IV'_{(1)}) \ldots (K'_{(n-1)}, IV'_{(n-1)})$ pairs.

For $n = 1$, a (K, IV) pair has $(K'_{(1)}, IV'_{(1)})$ that generates a only 1-bit shifted keystream with probability of 2^{-2}, which is greatly high probability compared to an ideal stream cipher that generates a random keystream by (K, IV). If an attacker can access to stream ciphers using such a slide pair, it is easy to distinguish keystreams from random streams.

Table 8.2: Examples of slide pairs

1-bit slide pair			
K	IV	$K'_{(1)}$	$IV'_{(1)}$
4bdf973abdd66263$_x$	49cba4aa656336eb$_x$	97bf2e757bacc4c7$_x$	93974954cac66dd7$_x$
d4ef3bfb30609c57$_x$	be0b3db8cc516480$_x$	a9de77f660c138af$_x$	7c167b7198a2c901$_x$
	95b8910812c5c95b$_x$		2b712210258b92b7$_x$
keystream		keystream	
001000110011010100110101100110112		010001100110101001101011001101112	
101000101111111001111001000010002		010001011111110011110010000100012	
5-bit slide pair			
K	IV	$K'_{(5)}$	$IV'_{(5)}$
5f3c0c948aa9262e$_x$	01660552b4169c5d$_x$	e78192915524c5cf$_x$	2cc0aa5682d38bba$_x$
7e55d395a458fba2$_x$	d584fedb576094f6$_x$	caba72b48b1f745b$_x$	b09fdb6aec129ed9$_x$
	cb5cd06e132a3644$_x$		6b9a0dc26546c889$_x$
keystream		keystream	
10010111000000001110011011101011$_2$		11100000000111001101110101101100$_2$	
01100011111110000011011001001111$_2$		01111111000001101100100111100110$_2$	
10-bit slide pair			
K	IV	$K'_{(10)}$	$IV'_{(10)}$
d048119b66a37d84$_x$	8b75aad54c32a2b6$_x$	20466d9a8df61354$_x$	d6ab5530ca8adbc4$_x$
d51287aef2f796d1$_x$	f118a4764dd0560a$_x$	4a1ebbcbde5b4738$_x$	6291d93741582a23$_x$
	88fc32827bc213cc$_x$		f0ca09ef084f334b$_x$
keystream		keystream	
00100010100010100100010110010111$_2$		00101001000101100101111001110010$_2$	
10011100100100101001110011010111$_2$		01001010011100110101110101001001$_2$	

8.4.3 Partial Slide Property of RAKAPOSHI

Here, we introduce a variant of the slide property. Recall that conditions 1-3 in Section 7.4 ensure that a (K, IV) pair has an n-bit slide pair $(K'_{(n)}, IV'_{(n)})$ that produces an n-bit sliding keystream of (K, IV). If the condition 3 does not hold, it is not ensured that $a_{448+128+i}$ and $a'_{448+128+i+n}$ are identical for $0 \leq i < n$, due to the difference of usage of s. However, these differences do no affect generations of $z_{n+1}(= z'_1), \ldots, z_{60}(= z'_{60-n})$. Thus, if only the conditions 1 and 2 hold, we can obtain the keystream pairs in which $\{z_{n+1}, \ldots, z_{60}\}$ and $\{z'_1, \ldots, z'_{60-n}\}$ are identical. We call such pair n-bit *partial slide pair*.

Therefore, a (K, IV) pair has an n-bit partial slide pair $(K''_{(n)}, IV''_{(n)})$ that generates an n-bit partial sliding keystream with probability of 2^{-n}, that occurs with higher probability than the basic slide property. For $n = 1$, a (K, IV) pair has a 1-bit partial slide pair $(K''_{(1)}, IV''_{(1)})$ with probability of 2^{-1}, where 59 bits of $\{z_2, \ldots, z_{60}\}$ and $\{z'_1, \ldots, z'_{59}\}$ are identical. If an attacker can access to stream ciphers using such a partial slide pair, it is easy to distinguish keystreams from random streams. The success probability is $1 - 2^{-59}$, because the 59-bit conditions of the keystream coincidentally hold with probability of 2^{-59} independently from the event of the partial slide pair.

8.5 Related-Key Attack on RAKAPOSHI

In this section, we give a technique for exploiting the slide property of RAKAPOSHI in order to construct a related-key key recovery attack on RAKAPOSHI. To begin with, we explain a method for determining a part of key bits by utilizing the 1-bit slide property. After that, we generalize it and propose a related-key attack on RAKAPOSHI based on the n-bit slide property. Moreover we improve it by using the partial slide property.

8.5.1 Related-Key Attacks Using 1-bit Slide Pair

Define the related key $K^*_{(1)}$ of this attack as [2]

$$K^*_{(1)} = \{k^*_0, k^*_1, \ldots, k^*_{127}\} = \{k_1, k_2, \ldots, k_{127}, x_0\}$$

where

$$\begin{aligned} x_0 &= g(a_0, a_6, a_7, a_{11}, a_{16}, a_{28}, a_{36}, a_{46}, a_{55}, a_{62}), \\ &= g(k_0, k_6, k_7, k_{11}, k_{16}, k_{28}, k_{36}, k_{46}, k_{55}, k_{62}). \end{aligned}$$

Since x_0 includes only key bits and does not depends on the value of IV, a related key K^* is determined if K is given. In the related-key setting, an attacker knows that a pair of $(K, K^*_{(1)})$ holds this relation, though actual values of those are unknown.

This attack uses chosen IV pairs (IV, $IV^*_{(1)}$) satisfying following relation,

$$\begin{aligned} IV &= \{iv_1, iv_2, \ldots, iv_{191}, y_0\} \\ &= \{iv^*_0, iv^*_1, \ldots, iv^*_{191}\} = IV^*_{(1)}, \end{aligned}$$

where

$$\begin{aligned} y_0 &= f(b_0, b_{14}, b_{37}, b_{41}, b_{49}, b_{51}, b_{93}, b_{107}, b_{120}, b_{134}, \\ &\quad b_{136}, b_{155}, b_{158}, b_{176}, a_{41}, a_{89}) \\ &\quad \oplus v(a_{67}, a_{127}, b_{23}, b_{53}, b_{77}, b_{81}, b_{103}, b_{128}) \\ &= f(iv_0, iv_{14}, iv_{37}, iv_{41}, iv_{49}, iv_{51}, iv_{93}, iv_{107}, iv_{120}, iv_{134}, \\ &\quad iv_{136}, iv_{155}, iv_{158}, iv_{176}, k_{41}, k_{89}) \\ &\quad \oplus v(k_{67}, k_{127}, iv_{23}, iv_{53}, iv_{77}, iv_{81}, iv_{103}, iv_{128}). \end{aligned}$$

In the chosen-IV setting, an attacker is able to choose the values of IV freely. Given IV, we can determined $IV^*_{(1)}$ except iv^*_{191} ($= y_0$), because y_0 includes four key bits, $\{k_{41}, k_{89}, k_{67}, k_{127}\}$, which are secret values even in the related-key setting.

If the value of iv^*_{191} is correctly guessed, (K, IV) and $(K^*_{(1)}, IV^*_{(1)})$ satisfy the condition 1 regarding the 1-bit slide pair. Then, $(K^*_{(1)}, IV^*_{(1)})$ generates a 1-bit shifted keystream of (K, IV) with probability of 2^{-2}. Since the probability that iv^*_{191} is correctly guessed is 2^{-1}, we expect to obtain one 1-bit sliding keystream pair after testing 2^3 $(IV, IV^*_{(1)})$ pairs. Once such a $(IV, IV^*_{(1)})$ pair is found, we

[2]This type of related keys has been utilized in attacks of Grain family [121, 122].

can confirm that $iv^*_{191}(= y_0)$ is correctly guessed. Then, a 1-bit equation of y_0, which includes 4 key bits of $\{k_{41}, k_{89}, k_{67}, k_{127}\}$, is obtained. Using four equations, $\{k_{41}, k_{89}, k_{67}, k_{127}\}$ can be determined with high probability.

The details of the attack procedure are given as follows:

1. Choose one pair of $(IV,\ IV^*_{(1)})$, where iv^*_{191} is guessed.
2. Obtain two keystreams of (K, IV) and $(K^*_{(1)}, IV^*_{(1)})$.
3. If these keystreams is the 1-bit sliding pair, then store the 1-bit equation of $\{k_{41}, k_{89}, k_{67}, k_{127}\}$ corresponding to iv^*_{191}.
4. Repeat step 1-3 until 4 equations are obtained.
5. Determine the key bits of $\{k_{41}, k_{89}, k_{67}, k_{127}\}$ by using four equations.
6. Obtain other 124 bits of the key in the brute force manner.

One equation can be obtained with probability of 2^{-3}. Thus, it is expected to repeat step 1-4 in 2^5 $(= 4 \times 2^3)$ times. The time complexity of the step 1-4 is 2^6 $(= 2 \times 2^5)$ initialization process, and the number of required chosen IVs is 2^6. In the step 5, we search $\{k_{41}, k_{89}, k_{67}, k_{127}\}$ by checking obtained 1-bit equations. Specifically, we guess values of $\{k_{41}, k_{89}, k_{67}, k_{127}\}$, and check whether these values satisfy all four equations. After testing 2^4 patterns of $\{k_{41}, k_{89}, k_{67}, k_{127}\}$, the one set that holds equations expects to be found. Thus, the time complexity of the step 5 is estimated as 2^4 estimations of equations, which is obviously less than 2^4 initialization process. Therefore, the whole time complexity is estimated as $2^{124}(\approx 2^4 + 2^6 + 2^{124})$ initialization process. This related-key attack recovers a key with time complexity of 2^{124}, 2^6 chosen IVs and one related key.

8.5.2 Related-Key Attacks Using n-bit Slide Pair

We extend the attack exploiting a 1-bit slide pair to an attack based on the n-bit slide pair. The related key $K^*_{(n)}$ and chosen IV pair are defined as,

$$\begin{aligned}
K^*_{(n)} &= \{k^*_0, k^*_1, \ldots, k^*_{127}\} = \{k_n, k_{n+1}, \ldots, k_{127}, x_0, \ldots, x_{n-1}\},\\
IV &= \{iv_n, iv_{n+1}, \ldots, iv_{191}, y_0, \ldots, y_{n-1}\}\\
&= \{iv^*_0, iv^*_1, \ldots, iv^*_{191}\} = IV^*_{(n)},
\end{aligned}$$

assuming that the values of n is less than 127. Table 3 shows involved key bits of each y_t for $0 \le t \le 12$.

If the values of $\{y_0, \ldots, y_{n-1}\}$ are correctly guessed with probability of 2^{-n}, $(K^*_{(n)}, IV^*_{(n)})$ generate an n-bit sliding keystream of (K, IV) with probability of 2^{-2n}. Once we find such pairs, n equations regarding each value of $\{y_0, \ldots, y_{n-1}\}$ are obtained. If y_t includes m bits of the key, m independent equations of y_t are needed for determining m bits of the key.

As an example, let us consider the attack using a 4-bit slide pair. $\{y_0, \ldots, y_3\}$ includes 4, 13, 13 and 13 key bits, respectively, and in total these involve independent 41 key bits. If 13 independent equations regarding each y are obtained[3], we can

[3]For y_0, 4 independent equations are enough.

Table 8.3: Included key bits in each y_t

y_t	Included key bits
y_0	41, 67, 89, 127
y_1	42, 68, 90, (0, 6, 7, 11, 16, 28, 36, 45, 55, 62)
y_2	43, 69, 91, (1, 7, 8, 12, 17, 29, 37, 46, 56, 63)
y_3	44, 70, 92, (2, 8, 9, 13, 18, 30, 38, 47, 57, 64)
y_4	45, 71, 93, (3, 9, 10, 14, 19, 31, 39, 48, 58, 65)
y_5	46, 72, 94, (4, 10, 11, 15, 20, 32, 40, 49, 59, 66)
y_6	47, 73, 95, (5, 11, 12, 16, 21, 33, 41, 50, 60, 67)
y_7	48, 74, 96, (6, 12, 13, 17, 22, 34, 42, 51, 61, 68)
y_8	49, 75, 97, (7, 13, 14, 18, 23, 35, 43, 52, 62, 69)
y_9	50, 76, 98, (8, 14, 15, 19, 24, 36, 44, 53, 63, 70)
y_{10}	51, 77, 99, (9, 15, 16, 20, 25, 37, 45, 54, 64, 71)
y_{11}	52, 78, 100, (10, 16, 18, 21, 26, 38, 46, 55, 65, 72)
y_{12}	53, 79, 101, (11, 17, 19, 22, 27, 39, 47, 56, 66, 73)

determine key bits included in each equation. It implies that this attack requires 13 pairs of $(IV, IV^*_{(4)})$ causing a 4-bit sliding keystream. These pairs are obtained with time complexity of 2^{17} $(= 13 \times 2 \times 2^{3\cdot 4})$ and $2^{17}(= 13 \times 2 \times 2^{3\cdot 4})$ chosen IVs. Then, 41 bits of the key can be determined with complexity of $2^{15}(= 2^4 + 2^{13} + 2^{13} + 2^{13})$ by exhaustively checking obtained equations. Therefore, the whole time complexity is estimated as $2^{87}(= 2^{87} + 2^{15} + 2^{17})$ initialization process. This related-key attack recovers the key with time complexity of 2^{87} and 2^{17} chosen IVs.

Using 8-bit slide pairs, each values of $\{y_0, \ldots, y_7\}$ includes 13 bits of the key except y_0 and in total these involve independent 75 key bits. 13 pairs of $(IV, IV^*_{(8)})$ causing a 8-bit sliding keystream are obtained with time complexity of 2^{29} $(= 13 \times 2 \times 2^{3\cdot 8})$ and $2^{29}(= 13 \times 2 \times 2^{3\cdot 8})$ chosen IVs. Then, in total 75 bits of the key are determined with complexity of $2^{16}(= 2^4 + 2^{13} \times 7)$ by exhaustively checking obtained equations. Therefore, the whole time complexity is estimated as $2^{54}(= 2^{53}+2^{16}+2^{29})$ initialization process. This related-key attack recovers the key with time complexity of 2^{54} and 2^{29} chosen IVs.

Using 11-bit slide pairs, each values of $\{y_0, \ldots, y_{10}\}$ includes 13 bits of the key except y_0 and in total these involve independent 88 key bits. 13 pairs of $(IV, IV^*_{(11)})$ causing a 11-bit sliding keystream are obtained with time complexity of 2^{38} $(= 13 \times 2 \times 2^{3\cdot 11})$ and $2^{38}(= 13 \times 2 \times 2^{3\cdot 11})$ chosen IVs. Then, in total 88 bits of the key are determined with complexity of $2^{17}(= 2^4 + 2^{13} \times 10)$ by exhaustively checking obtained equations. Therefore, the whole time complexity is estimated as $2^{41}(= 2^{40} + 2^{17} + 2^{38})$ initialization process. This related-key attack recovers the key with time complexity of 2^{41} and 2^{38} chosen IVs. This attack is optimized for the time complexity.

Table 8.4: Summary of our related-key attacks

Used Slide pair	Time Complexity	Chosen IVs
1 bit	2^{124}	2^{6}
4 bit	2^{87}	2^{17}
8 bit	2^{54}	2^{29}
11 bit	2^{41}	2^{38}
1 bit (partial)	2^{124}	2^{6}
4 bit (partial)	2^{87}	2^{13}
8 bit (partial)	2^{54}	2^{21}
11 bit (partial)	2^{41}	2^{27}
13 bit (partial)	2^{33}	2^{30}

8.5.3 Related-Key Attacks Using n-bit Partial Slide Pair

This attack is further improved by using n-bit partial slide pair in which $\{z_{n+1}, \ldots, z_{60}\}$ and $\{z'_1, \ldots, z'_{60-n}\}$ of keystreams are identical. Using 13-bit partial slide pairs, i.e., 47 bits of $\{z_{14}, \ldots, z_{60}\}$ and $\{z'_1, \ldots, z'_{47}\}$ are identical, each values of $\{y_0, \ldots, y_{12}\}$ includes 13 bits of the key except y_0 and in total these involve independent 96 key bits. 13 pairs of $(IV, IV^*_{(13)})$ causing a 13-bit partial sliding keystream are obtained with time complexity of 2^{30} $(= 13 \times 2 \times 2^{2 \cdot 13})$ and $2^{30}(= 13 \times 2 \times 2^{2 \cdot 13})$ chosen IVs. Then, in total 96 bits of the key are determined with complexity of $2^{18}(= 2^4 + 2^{13} \times 12)$ by exhaustively checking obtained equations. Therefore, the whole time complexity is estimated as $2^{33}(= 2^{32} + 2^{18} + 2^{30})$ initialization process. This related-key attack recovers the key with time complexity of 2^{33} and 2^{30} chosen IVs. The success probability is estimated as $(1 - 2^{-47})^{13} \approx 1$.

This technique also allow to improve the attacks based on 1, 4, 8 and 11 -bit slide pairs with respect to data complexity. Table 4 shows the summary of our results. This result reveals that RAKAPOSHI is practically vulnerable to the related-key attack based on the slide property.

8.6 Speed-up Keysearch on RAKAPOSHI

In this section, we give a method for speeding up a keysearch in the single key setting.

In order to improve the naive brute force attack, we exploit partial slide pairs. In particular, we utilize the observation that if the condition 2 regarding n-bit partial slide pairs holds, we can check n keys without recalculations of the initialization process.

Assume that an attacker aims to find K^{target} in the brute-force style search, i.e., test all keys with the keystream of $(K^{target}, IV^{target})$. Let us consider that a candidate pair of (K, IV) is set for the test. In the initialization process of (K, IV), if $s^{320+i} = 0$ (condition 2) holds for $0 \leq i < n$, then (K, IV) surely has $1, \ldots, n$ bit partial slide pairs such that $\{(K_{(1)}, IV_{(1)}), \ldots (K_{(n)}, IV_{(n)})\} = \{(A^1, B^1), \ldots (A^n, B^n)\}$.

Then we can simultaneously verify n keys with only initialization call of (K, IV) by using additional keystreams of $\{(K^{target}, IV_{(1)}), \ldots (K^{target}, IV_{(n)})\}$. Note that n bits of $IV_{(n)}$, namely $y_0, \ldots, y_n$, are uncontrollable and can not be fixed, while other $(192 - n)$ bits of $IV_{(n)}$ is determined from IV^{target}. Thus, this attack requires a set of keystreams generated from $1 + 2^1 + 2^2 \ldots + 2^n$ chosen IVs.

The detailed algorithm is as follows:

1. Set $K = 0$ and $IV = IV^{target}$
2. Perform the initialization process and generate keystream bits $(z_0 \ldots z_{60})$.
3. For $t = 0$ to [smallest $0 \leq n < 60$ for which $s^{320+n+1} = 1$], check $(z_t \ldots z_{60})$ match the keystream of $(K^{Target}, IV_{(t)})$.
    - if matching, output $K^{target} = A^t$
4. Updating $K = A^{t+1}$, and Return step 2 only if $K \neq 0$.

As estimated in [122], this algorithm will eventually reach $K = 0$ again, because K is updated in the invertible way. Then, it is expected that this code check 2^{127} key values. The expected number of checked values of K in the step 3 for each loop of step 2-4 is 2 ($\approx 1 + 1 \cdot 1/4 + 2 \cdot 1/8 + \ldots$). Thus, the complexity of this algorithm is estimated as 2^{126} initialization processes of the step 2. If we can not find the target key, the algorithm can be repeated with different starting values which have a different cycle.

In order to estimate the actual cost of the attack, we consider the case where $1 - 10$ bits partial slide pairs are used in this algorithm. Since the expected number of checked values of K in the step 3 is 2 ($\approx 1+1\cdot 1/4+2\cdot 1/8+\ldots+10\cdot 1/1024$), time complexity for searching 2^{127} key values is estimate 2^{126}. After that, to cover all key space, we will check another cycle with same complexity. The whole complexity is given as 2^{127} initialization processes. The number of the set of IV used for the attack is 2^{11} ($\approx 1 + 2 + 2^2 + \ldots + 2^{10}$).

Therefore, we can speed up keysearch by a factor two. In the Grain v1 attack [122], this type of attack seems applicable in the case of that IV are all 1. Unlike the attack on Grain v1, our attack on RAKAPOSHI can be done for any IV while a set of chosen IVs are needed. This shows that RAKAPOSHI has a 127-bit security instead of 128 bits. However, this attack is a marginal improvement compared to the brute force attack. Thus, we do not claim this to be real attack based on algorithmic weakness.

8.7 Conclusion

this chapter has investigated slide properties of RAKAPOSHI stream ciphers. First, we have shown that for RAKAPOSHI, any Key-IV pair has a corresponding *slide* Key-IV pair that generates a n-bit shifted keystream with probability of 2^{-2n}. Then, we showed that this property is able to be converted into key recovery attacks on RAKAPOSHI. In the related-key setting, our attack based on the slide property can recover a 128-bit key with time complexity of 2^{33} and 2^{30} chosen IVs. In addition, a method for speeding up the brute force attack by a factor of 2 can be constructed in the single key setting.

These results mainly exploit the self-similarity of the state update function of RAKAPOSHI. If the self-similarity is destroyed, this type of attack can be avoid. For example, inserting round constants or a counter value in each step is effective for preventing the attack presented in this chapter.

Chapter 9

Cryptanalysis of RC4

This chapter investigates the practical security of RC4 in broadcast setting where the same plaintext is encrypted with different user keys. We introduce several new biases in the initial (1st to 257th) bytes of the RC4 keystream, which are substantially stronger than known biases. Combining the new biases with the known ones, a cumulative list of strong biases in the first 257 bytes of the RC4 keystream is constructed. We demonstrate a plaintext recovery attack using our strong bias set of initial bytes by the means of a computer experiment. Almost all of the first 257 bytes of the plaintext can be recovered, with probability more than 0.8, using only 2^{32} ciphertexts encrypted by randomly-chosen keys. We also propose an efficient method to extract later bytes of the plaintext, after the 258th byte. The proposed method exploits our bias set of first 257 bytes in conjunction with the digraph repetition bias proposed by Mantin in EUROCRYPT 2005, and sequentially recovers the later bytes of the plaintext after recovering the first 257 bytes. Once the possible candidates for the first 257 bytes are obtained by our bias set, the later bytes can be recovered from about 2^{34} ciphertexts with probability close to 1.

Finally, we show that our set of these biases are applicable to plaintext recovery attacks, key recovery attacks and distinguishing attacks.

9.1 Introduction

RC4, designed by Rivest in 1987, is one of most widely used stream ciphers in the world. It is adopted in many software applications and standard protocols such as SSL/TLS, WEP, Microsoft Lotus and Oracle secure SQL. RC4 consists of a key scheduling algorithm (KSA) and a pseudo-random generation algorithm (PRGA). The KSA converts a user-provided variable-length key (typically, 5–32 bytes) into an initial state S consisting of a permutation of $\{0, 1, 2, \ldots, N-1\}$, where N is typically 256. The PRGA generates a keystream Z_1, Z_2, …, Z_r, … from S, where r is a round number of the PRGA. Z_r is XOR-ed with the r-th plaintext byte P_r to obtain the ciphertext byte C_r. The algorithm of RC4 is shown in Algorithm 2, where $+$ denotes arithmetic addition modulo N, ℓ is the key length, and i and j are used to point to the locations of S, respectively. Then, $S[x]$ denotes the value of S indexed x.

After the disclosure of its algorithm in 1994, RC4 has attracted intensive cryptanalytic efforts over past 20 years. Distinguishing attacks, which attempt to distin-

Algorithm 2 RC4 Algorithm

```
KSA(K):
  for $i = 0$ to $N - 1$ do
    $S[i] \leftarrow i$
  end for
  $j \leftarrow 0$
  for $i = 0$ to $N - 1$ do
    $j \leftarrow j + S[i] + K[i \bmod \ell]$
    Swap $S[i]$ and $S[j]$
  end for

PRGA(K):
  $i \leftarrow 0$
  $j \leftarrow 0$
  $S \leftarrow KSA(K)$
  loop
    $i \leftarrow i + 1$
    $j \leftarrow j + S[i]$
    Swap $S[i]$ and $S[j]$
    Output $Z \leftarrow S[S[i] + S[j]]$
  end loop
```

guish an RC4 keystream from a random stream, were proposed in [123, 124, 37, 33, 125, 126, 34]. State recovery attack, which recovers a full state instead of the user-provided key, was shown by Knudsen et al. [127], and it was improved by Maximov and Khovratovich [128]. Other types of attacks are also proposed, e.g., key collision attack [129], keystream predictive attack [37] and key recovery attacks from a state [130, 131].

In FSE 2001, Mantin and Shamir presented an attack on RC4 in the broadcast setting where the same plaintext is encrypted with different user keys [33]. The Mantin-Shamir attack can extract the second byte of the plaintext from only $\Omega(N)$ ciphertexts encrypted with randomly-chosen different keys by exploiting a bias of Z_2. Specifically, the event $Z_2 = 0$ occurs with twice the expected probability of a random one. In FSE 2011, Maitra, Paul and Sen Gupta showed that $Z_3, Z_4, \ldots, Z_{255}$ are also biased to 0 [34]. Then the bytes 3 to 255 can also be recovered in the broadcast setting, from $\Omega(N^3)$ ciphertexts.

Although the broadcast attacks were theoretically estimated, we find that three questions are still open in terms of a practical security of broadcast RC4.

1. *Are the biases exploited in the previous attacks the strongest biases for the initial bytes 1 to 255?*
2. *While the previous results [33, 34] estimate only lower bounds* (Ω)*, how many ciphertexts encrypted with different keys are actually required for a practical attack on broadcast RC4?*
3. *Is it possible to efficiently recover the later bytes of the plaintext, after byte 256?*

9.1.1 Our Contribution

In this chapter, we provide answers to all the aforesaid questions. To begin with, we introduce a new bias regarding Z_1, which is a conditional bias such that Z_1 is biased to 0 when Z_2 is 0. Using this bias in conjunction with the bias of $Z_2 = 0$ [33], the first byte of a plaintext is extracted from $\Omega(N^2)$ ciphertexts encrypted with different keys. Although the strong bias of the first byte, which is a negative bias towards zero, has already been pointed out in [125, 35], it requires $\Omega(N^3)$ ciphertexts to extract the first byte of the plaintext. Thus, the new conditional bias observed by us is very useful, because the number of required ciphertexts to recover the first byte reduces by a factor of $N/2$ compared the straightforward method. Besides, we introduce new strong biases, i.e., $Z_3 = 131$, $Z_r = r$ for $3 \leq r \leq 255$, and extended keylength-dependent biases such that $Z_{x \cdot \ell} = -x \cdot \ell$ for $x = 2, 3, \ldots, 7$ and $\ell = 16$, which are extensions of the keylength-dependent biases in which only the parameter of $x = 1$ is considered [36]. These new biases are substantially stronger than known biases of $Z_r = 0$ in case of certain bytes within $Z_3, Z_4, \ldots, Z_{255}$. After providing theoretical considerations for these biases, we experimentally confirm the validity of the same. Combining the new biases with known biases, we construct a cumulative list of strongest known biases in $Z_1, Z_2, \ldots, Z_{255}$. At the same time, we experimentally show two new biases of Z_{256} and Z_{257}, and add these to our bias set. Note that biases of $Z_2, Z_3, \ldots, Z_{257}$ included in our bias set are *strongest* biases amongst all *single* positive and negative biases of each byte when a 16-byte (128-bit) key is used.

We demonstrate a plaintext recovery attack using our bias set by the computer experiment, and estimate the number of required ciphertexts and success probability when $N = 256$. Almost all first 257 bytes, $P_1, P_2, \ldots, P_{257}$, can be extracted with probability more than 0.8 from 2^{32} ciphertexts encrypted by randomly-chosen keys. Given 2^{34} ciphertexts, all bytes of $P_1, P_2, \ldots, P_{257}$ can be narrowed down to two candidates each with probability one. This is a first practical security evaluation of broadcast RC4 using all known biases of the cipher, and some new ones that we observe.

Finally, an efficient method to extract later bytes of the plaintext, namely bytes after P_{258}, is given. It exploits our bias set of $Z_1, Z_2, \ldots, Z_{257}$ in conjunction with the digraph repetition bias proposed by Mantin [37], and then sequentially recovers bytes of the plaintext. Once the possible candidates for $P_1, P_2, \ldots, P_{257}$ are obtained by our bias set, P_r ($r \geq 258$) are recovered from about 2^{34} ciphertexts with probability one. Since the digraph repetition bias is a long-term bias, which occurs in any keystream byte, our sequential method is expected to recover any plaintext byte from only ciphertexts produced by different randomly-chosen keys. We show that the first 2^{50} bytes $\approx$ 1000 T bytes of the plaintext can be recovered from 2^{34} ciphertexts with probability of 0.97170.

Also, the broadcast setting is converted into the multi-session setting of SSL/TLS where the target plaintext block are repeatedly sent in the same position in the plaintexts in multiple sessions.

9.2 Known Attacks on Broadcast RC4

This section briefly reviews known attacks on RC4 in the broadcast setting where the same plaintext is encrypted with different randomly-chosen keys.

9.2.1 Mantin-Shamir (MS) Attack

Mantin and Shamir first presented a broadcast RC4 attack exploiting a bias of Z_2 [33].

Theorem 3. **[33]** *Assume that the initial permutation S is randomly chosen from the set of all the possible permutations of $\{0, 1, 2, \ldots, N-1\}$. Then the probability that the second output byte of RC4 is 0 is approximately $\frac{2}{N}$.*

This probability is estimated as $\frac{2}{256}$ when $N = 256$. Based on this bias, the broadcast RC4 attack is demonstrated by Theorems 14 and 5.

Theorem 4. **[33]** *Let X and Y be two distributions, and suppose that the event e happens in X with probability p and in Y with probability $p \cdot (1+q)$. Then for small p and q, $\mathrm{O}(\frac{1}{p \cdot q^2})$ samples suffice to distinguish X from Y with a constant probability of success.*

In this case, p and q are given as $p = 1/N$ and $q = 1$. The number of samples is about N.

Theorem 5. **[33]** *Let P be a plaintext, and let $C^{(1)}, C^{(2)}, \ldots, C^{(k)}$ be the RC4 encryptions of P under k uniformly distributed keys. Then, if $k = \Omega(N)$, the second byte of P can be reliably extracted from $C^{(1)}, C^{(2)}, \ldots, C^{(k)}$.*

According to the relation $C_2^{(i)} = P_2^{(i)} \oplus Z_2^{(i)}$, if $Z_2^{(i)} = 0$ holds, then $C_2^{(i)}$ is same as $P_2^{(i)}$. From Theorem 3, $Z_2 = 0$ occurs with twice the expected probability of a random one. Thus, most frequent byte in amongst $C_2^{(1)}, C_2^{(2)}, \ldots, C_2^{(k)}$ is likely to be P_2 itself. When $N = 256$, it requires more than 2^8 ciphertexts encrypted with randomly-chosen keys.

9.2.2 Maitra, Paul and Sen Gupta (MPS) Attack

Maitra, Paul and Sen Gupta showed that $Z_3, Z_4, \ldots, Z_{255}$ are also biased to 0 [34, 35]. Although the MS attack assumes that an initial permutation S is random, the MPS attack exploits biases of S after the KSA [132]. Let $S_r[x]$ be the value of S indexed x after r round, where S_0 is the initial state of RC4 after the KSA. Biases of the initial state of the PRGA are given as follow.

Proposition 1. [132] *After the end of KSA, for $0 \le u \le N-1, 0 \le v \le N-1$,*

$$\Pr(S_0[u] = v) = \begin{cases} \frac{1}{N} \cdot \left((\frac{N-1}{N})^v + (1 - (\frac{N-1}{N})^v) \cdot (\frac{N-1}{N})^{N-u-1} \right) & (v \le u), \\ \frac{1}{N} \cdot \left((\frac{N-1}{N})^{N-u-1} + (\frac{N-1}{N})^v \right) & (v > u). \end{cases}$$

The probability of $S_{r-1}[r]$ in the PRGA are given as the follows.

Theorem 6. **[35]** [1] *For $3 \le r \le N-1$, the probability $\Pr(S_{r-1}[r] = v)$ is approximately*

$$\Pr(S_1[r] = v) \cdot \left(1 - \frac{1}{N}\right)^{r-2} + \sum_{t=2}^{r-1} \sum_{w=0}^{r-t} \frac{\Pr(S_1[t] = v)}{w! \cdot N} \cdot \left(\frac{r-t-1}{N}\right)^w \cdot \left(1 - \frac{1}{N}\right)^{r-3-w},$$

where $\Pr(S_1[t] = v)$ is given as

$$\Pr(S_1[t] = v) = \begin{cases} \Pr(S_0[1] = 1) + \sum_{X \ne 1} \Pr(S_0[1] = X \wedge S_0[X] = 1) & (t = 1, v = 1), \\ \sum_{X \ne 1, v} \Pr(S_0[1] = X \wedge S_0[X] = v) & (t = 1, v \ne 1), \\ \Pr(S_0[1] = t) + \sum_{X \ne t} \Pr(S_0[1] = X \wedge S_0[t] = t) & (t \ne 1, v = t), \\ \sum_{X \ne t, v} \Pr(S_0[1] = X \wedge S_0[t] = v) & (t \ne 1, v \ne t). \end{cases}$$

[1]The theorems with respect to $Z_r = 0$ in [34] and [35] are slightly different. this chapter uses the results from the full version [35].

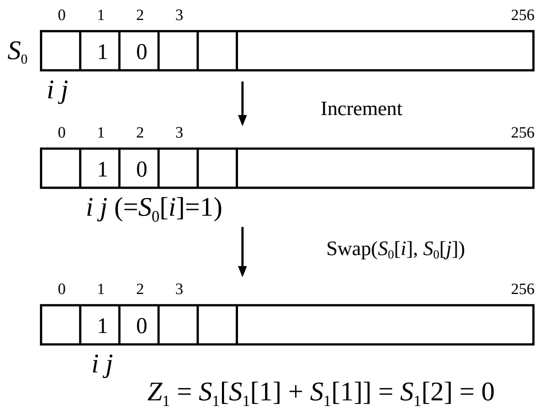


Figure 9.1: Event for bias of $Z_1 = 0|Z_2 = 0$

Then, the bias of $\Pr(Z_r = 0)$ is estimated as follows.

Theorem 7. **[35]** *For* $3 \leq r \leq N-1$, $\Pr(Z_r = 0)$ *is approximately*

$$\Pr(Z_r = 0) \approx \frac{1}{N} + \frac{c_r}{N^2},$$

where c_r *is given as*

$$c_r = \begin{cases} \frac{N}{N-1} \cdot (N \cdot \Pr(S_{r-1}[r] = r) - 1) - \frac{N-2}{N-1} & (r = 3), \\ \frac{N}{N-1} \cdot (N \cdot \Pr(S_{r-1}[r] = r) - 1) & (r \neq 3). \end{cases}$$

Since the parameters of p and q are given as $p = 1/N$ and $q = c_r/N$, The number of required ciphertexts with different keys for the extraction of $P_3, P_4, \ldots, P_{255}$ is roughly estimated as $\Omega(N^3)$.

9.3 New Biases : Theory and Experiment

This section introduces four new biases in the keystream of RC4. To begin with, we prove a conditional bias of Z_1 towards 0 when $Z_2 = 0$. After that, we present new biases in the events, $Z_3 = 131$, $Z_r = r$, and extended keylength-dependent biases, which are substantially stronger than the known biases such as $Z_r = 0$. Then, we construct a cumulative list of strong biases in $Z_1, Z_2, \ldots, Z_{257}$ to mount an efficient plaintext recovery attack on broadcast RC4.

9.3.1 Bias of $Z_1 = 0|Z_2 = 0$

A new conditional bias such that Z_1 is biased to 0 when $Z_2 = 0$ is given as Theorem 8.

Theorem 8. $\Pr(Z_1 = 0|Z_2 = 0)$ *is approximately*

$$\Pr(Z_1 = 0|Z_2 = 0) \approx \frac{1}{2} \cdot \left(\Pr(S_0[1] = 1) + (1 - \Pr(S_0[1] = 1)) \cdot \frac{1}{N}\right) + \frac{1}{2} \cdot \frac{1}{N}.$$

Proof. Two cases of $S_0[2] = 0$ and $S_0[2] \neq 0$ are considered. As mentioned in [33], when Z_2 is 0, $S_0[2]$ is also 0 with probability of $\frac{1}{2}$.

- $S_0[2] = 0$
  For $i = 1$, if $S_0[1]$ is 1, the index j is updated as $j = S_0[i] = S_0[1] = 1$. Then the first output byte Z_1 is expressed as follows (see Fig. 9.1),
  $$Z_1 = S_1[S_1[i] + S_1[j]] = S_1[S_1[1] + S_1[1]] = S_1[2] = S_0[2] = 0.$$
  Assuming that $Z_1 = 0$ holds with probability of $\frac{1}{N}$ when $S_0[1] \neq 1$, the probability of $\Pr(Z_1 = 0|S_0[2] = 0)$ is estimated as
  $$\Pr(Z_1 = 0|S_0[2] = 0) = \Pr(S_0[1] = 1) + (1 - \Pr(S_0[1] = 1)) \cdot \frac{1}{N}.$$
- $S_0[2] \neq 0$
  Suppose that the event of $Z_1 = 0$ occurs with probability of $\frac{1}{N}$. Then $\Pr(Z_1 = 0|S_0[2] = 0)$ is estimated as
  $$\Pr(Z_1 = 0|S_0[2] \neq 0) = \frac{1}{N}.$$

Therefore $\Pr(Z_1 = 0|Z_2 = 0)$ is approximately

$$\begin{aligned}\Pr(Z_1 = 0|Z_2 = 0) &= \Pr(Z_1 = 0|S_0[2] = 0) \cdot \Pr(S_0[2] = 0|Z_2 = 0) \\ &\quad + \Pr(Z_1 = 0|S_0[2] \neq 0) \cdot \Pr(S_0[2] \neq 0|Z_2 = 0) \\ &\approx \frac{1}{2} \cdot \left(\Pr(S_0[1] = 1) + (1 - \Pr(S_0[1] = 1)) \cdot \frac{1}{N}\right) + \frac{1}{2} \cdot \frac{1}{N}.\end{aligned}$$

□ □

When $N = 256$, $\Pr(S_0[1] = 1)$ is obtained by Proposition 1.

$$\Pr(S_0[1] = 1) = \frac{1}{256} \cdot \left(\left(\frac{255}{256}\right) + \left(1 - \left(\frac{1}{256}\right)\right) \cdot \left(\frac{1}{256}\right)^{254}\right) = 0.0038966.$$

Then, $\Pr(Z_1 = 0|Z_2 = 0)$ is computed as

$$\begin{aligned}\Pr(Z_1 = 0|Z_2 = 0) &= \frac{1}{2} \cdot \left(\Pr(S_0[1] = 1) + (1 - \Pr(S_0[1] = 1)) \cdot \frac{1}{256}\right) + \frac{1}{2} \cdot \frac{1}{256} \\ &= 0.0058470 = 2^{-7.418} = 2^{-8} \cdot (1 + 2^{-1.009}).\end{aligned}$$

Since the experimental value of $\Pr(Z_1 = 0|Z_2 = 0)$ for 2^{40} randomly-chosen keys is obtained as $0.0058109 = 2^{-8} \cdot (1 + 2^{-1.036})$, the theoretical value is correctly approximated.

From this bias, $\Pr(Z_1 = 0 \wedge Z_2 = 0)$ can also be estimated, as follows.

$$\Pr(Z_1 = 0 \wedge Z_2 = 0) = \Pr(Z_2 = 0) \cdot \Pr(Z_1 = 0|Z_2 = 0).$$

When $N = 256$, it is estimated as

$$\Pr(Z_1 = 0 \wedge Z_2 = 0) = \frac{2}{256} \cdot 2^{-7.418} = 2^{-14.418} = 2^{-16} \cdot (1 + 2^{0.996}).$$

This type of bias, called digraph bias, was proved as a long term bias by Fluhrer and McGrew [124]. However, such a strong bias in initial bytes was not reported. Specifically, the probability of the general long-term digraph bias is estimated as $2^{-16} \cdot (1 + 2^{-8})$ in [124] when $N = 256$, while that of our bias is $2^{-16} \cdot (1 + 2^{0.996})$. Thus our result reveals that the digraph bias in initial bytes is much stronger than what is estimated in [124].

Note that we searched for the similar form of conditional biases in first 256 bytes of the RC4 keystream. In particular, we check following specific patterns, $(Z_{r-a} = X | Z_r = Y)$ for $0 \leq X, Y \leq 255$, $2 \leq r \leq 256$, $1 \leq a \leq 8$. However, such a strong bias could not be found in our experiment, while all conditional biases are not covered.

Application to Broadcast RC4 attack:

Using this new conditional bias of $Z_1 = 0 | Z_2 = 0$ in conjunction with the bias of $Z_2 = 0$ [33], the first byte of the plaintext can be efficiently extracted, where $N = 256$. After 2^{17} ciphertexts with randomly-chosen keys are collected, following procedures are performed.

Step 1 Extract the second byte of the target plaintext, P_2, from 2^8 ciphertexts [33].

Step 2 Find the ciphertext in which $Z_2 = 0$ is XOR-ed by the computation of $C_2 \oplus P_2$. Then, $2^{10} = 2^{17} \cdot 2/256$ ciphertexts matching this criterion are expected to be obtained.

Step 3 Regard the most frequent byte in the first byte C_1 of these matching 2^{10} ciphertexts as P_1.

In Step 3, using the bias of $\Pr(Z_1 = 0 | Z_2 = 0) = 2^{-8} \cdot (1 + 2^{-1.009})$, P_1 is extracted from remaining $2^{10} (\sim \frac{1}{2^{-8} \cdot (2^{-1.009})^2})$ ciphertexts by Theorems 14 and 5, assuming the relation of $C_1 = P_1 \oplus Z_1 = P_1$ holds. Although the bias of the first byte has already been pointed out in [125, 35], it requires 2^{24} ciphertexts to extract the first byte using the known biases, because the probability of the strongest bias, which is a negative bias of Z_1 towards 0, is estimated as about $2^{-8} \cdot (1 - 2^{-8})$ [35]. Thus, the new conditional bias identified by us is very efficient, because the number of required ciphertexts reduces by a factor close to $N/2$ compared to that of the straightforward method.

9.3.2 Bias of $Z_3 = 131$

A new bias of $Z_3 = 131$, which is stronger than $Z_3 = 0$ [34, 35], is given as Theorem 9.

Theorem 9. $\Pr(Z_3 = 131)$ *is approximately*

$$\begin{aligned}\Pr(Z_3 = 131) \quad \approx \quad & \Pr(S_0[1] = 131) \cdot \Pr(S_0[2] = 128) + \\ & (1 - \Pr(S_0[1] = 131) \cdot \Pr(S_0[2] = 128)) \cdot 1/N.\end{aligned}$$

Proof. Suppose the events $S_0[1] = 131$ and $S_0[2] = 128$ occur after the KSA. For $i = 1$, j is updated as $S_0[1] = 131$. After $S_0[1]$ and $S_0[131]$ are swapped, $S_1[131]$ becomes 131. For $i = 2$, j is updated as $131 + S_1[2] = 131 + S_0[2] = 131 + 128 =$

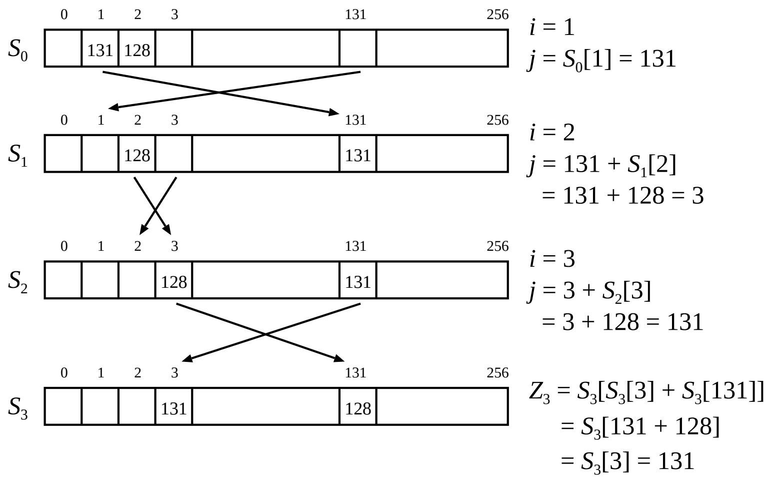


Figure 9.2: Event for bias of $Z_3 = 131$

3, and $S_1[2]$ and $S_1[3]$ are swapped. Then $S_2[3] = 128$ is obtained. Finally, for $i = 3$, j is updated as $3 + S_2[3] = 3 + 128 = 131$. After $S_2[3]$ and $S_2[131]$ are swapped, $S_3[3] = 131$ and $S_3[131] = 128$ holds. Then, a third output byte Z_3 is $Z_3 = S_3[S_3[3] + S_3[131]] = S_3[131 + 128] = S_3[3] = 131$. Thus, when $S_0[1] = 131$ and $S_0[2] = 128$ hold, $Z_3 = 131$ holds with probability one. Figure 9.2 depicts this event.

Assuming that in other cases, that is when $S_0[1] \neq 131$ or $S_0[2] \neq 128$, the event $Z_3 = 131$ holds with probability of $1/N$, the probability of $\Pr(Z_3 = 131)$ is estimated as

$$\begin{aligned}\Pr(Z_3 = 131) \;\approx\; & \Pr(S_0[1] = 131) \cdot \Pr(S_0[2] = 128) + \\ & (1 - \Pr(S_0[1] = 131) \cdot \Pr(S_0[2] = 128)) \cdot 1/N.\end{aligned}$$

□ □

When $N = 256$, by Proposition 1, $\Pr(S_0[1] = 131)$ and $\Pr(S_0[2] = 128)$ are estimated as

$$\Pr(S_0[1] = 131) = \frac{1}{256} \cdot \left(\left(\frac{255}{256}\right)^{256-1-1} + \left(\frac{255}{256}\right)^{131}\right) = 0.0037848,$$

$$\Pr(S_0[2] = 128) = \frac{1}{256} \cdot \left(\left(\frac{255}{256}\right)^{256-2-1} + \left(\frac{255}{256}\right)^{128}\right) = 0.0038181.$$

Thus, $\Pr(Z_r = 131)$ is computed as

$$\Pr(Z_3 = 131) \approx 0.0039206 = 2^{-8} \cdot (1 + 2^{-8.089}).$$

Since experimental value of this bias for 2^{40} randomly-chosen keys is obtained as $0.0039204 = 2^{-8} \cdot (1 + 2^{-8.109})$, the theoretical value is correctly approximated.

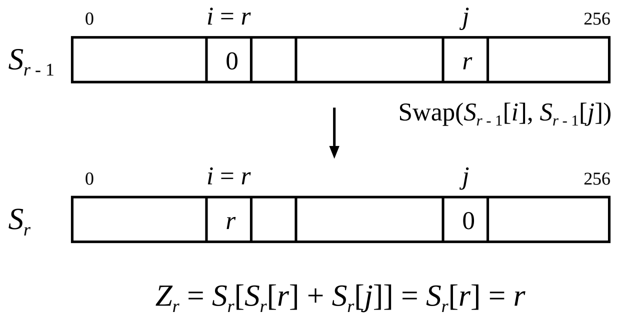

0
i = r
j
256
S_{r-1}
0
r
Swap(S_{r-1}[i], S_{r-1}[j])
0
i = r
j
256
S_r
r
0
Z_r = S_r[S_r[r] + S_r[j]] = S_r[r] = r

Figure 9.3: Event (Case 1) for bias of $Z_r = r$

Let us compare it to the bias of $Z_3 = 0$ of the MPS attack [34, 35]. The experimental value for 2^{40} randomly-chosen keys is obtained as

$$\Pr(Z_3 = 0) = 0.0039116 = 2^{-8} \cdot (1 + 2^{-9.512}).$$

Thus, the bias of $Z_3 = 131$ is stronger than that of $Z_3 = 0$.

We should utilize $Z_3 = 131$ instead of $Z_3 = 0$ for the efficient plaintext recovery attack. When $Z_3 = 131$ and $Z_3 = 0$ are jointly used, two candidates of P_3 remain. Thus, in order to detect one correct value of P_3, the only use of $Z_3 = 131$ is more efficient.

9.3.3 Bias of $Z_r = r$ for $3 \leq r \leq N - 1$

We also present a new bias in the event $Z_r = r$ for $3 \leq r \leq N-1$, whose probabilities are very close to those of $Z_r = 0$ [34], and the new biases are stronger than those of $Z_r = 0$ in some rounds. Thus, for an efficient attack, we need to carefully consider which biases are stronger in each round. The probability of $Z_r = r$ is given as Theorem 10.

Theorem 10. $\Pr(Z_r = r)$ *for* $3 \leq r \leq N - 1$ *is approximately*

$$\begin{aligned} \Pr(Z_r = r) \ &\approx \ p_{r-1,0} \cdot \frac{1}{N} + p_{r-1,r} \cdot \frac{1}{N} \cdot \frac{N-2}{N} + \\ &\quad (1 - p_{r-1,0} \cdot \frac{1}{N} - p_{r-1,r} \cdot \frac{1}{N} - (1 - p_{r-1,0}) \cdot \frac{1}{N} \cdot 2) \cdot \frac{1}{N}, \end{aligned}$$

where $p_{r-1,0} = \Pr(S_{r-1}[r] = 0)$ *and* $p_{r-1,r} = \Pr(S_{r-1}[r] = r)$.

Proof. Let i_r and j_r be r-th i and j, respectively. For $i_r = r$, an output Z_r is expressed as

$$Z_r \ = \ S_r[S_r[i_r] + S_r[j_r]] = S_r[S_r[r] + S_{r-1}[r]].$$

Then, let us consider four independent cases.

Case 1 : $S_{r-1}[r] = 0 \wedge S_r[r] = r$

Case 2 : $S_{r-1}[r] = r \wedge S_r[r] = j_r - r \wedge j_r \neq r, r + r$

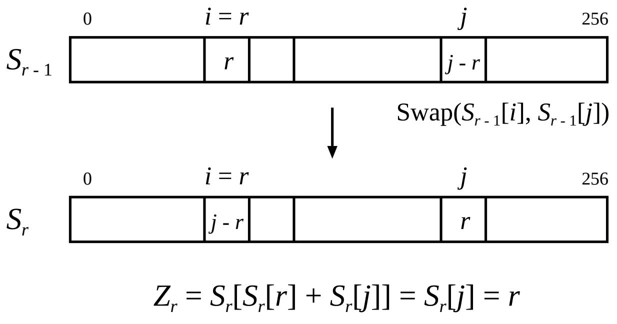

0
i = r
j
256
S_{r-1}
r
j - r
Swap($S_{r-1}[i]$, $S_{r-1}[j]$)
0
i = r
j
256
S_r
j - r
r
$Z_r = S_r[S_r[r] + S_r[j]] = S_r[j] = r$

Figure 9.4: Event (Case 2) for bias of $Z_r = r$

Case 3 : $S_{r-1}[r] \neq 0 \wedge S_r[r] = r - S_{r-1}[r]$

Case 4 : $S_{r-1}[r] \neq 0 \wedge S_r[r] = r$

In Case 1 and Case 2, the output is always $Z_r = r$. On the other hand, in Case 3 and Case 4, the output is not $Z_r = r$.

Case 1 : $S_{r-1}[r] = 0 \wedge S_r[r] = r$
The output is expressed as $Z_r = S_r[S_r[r] + S_{r-1}[r]] = S_r[r + 0] = S_r[r] = r$ (see Fig. 9.3). Then, the probability of $Z_r = r$ is one. Here $S_r[r]$ is chosen by pointer j. Since j_r for $r \geq 3$ behaves randomly [34], $S_r[r]$ is assumed to be uniformly random. it is estimated as

$$\Pr(S_{r-1}[r] = 0 \wedge S_r[r] = r) = p_{r-1,0} \cdot \frac{1}{N}.$$

Case 2 : $S_{r-1}[r] = r \wedge S_r[r] = j_r - r \wedge j_r \neq r, r + r$
The output is expressed as $Z_r = S_r[S_r[r] + S_{r-1}[r]] = S_r[j_r - r + r] = S_r[j_r] = S_{r-1}[r] = r$ (see Fig. 9.4). Then, the probability of $Z_r = r$ is one. Similar to Case 1, $S_r[r]$ is assumed to be uniformly random.

When $j_r = r$, the probability of $Z_r = r$ is zero because of the relation of $Z_r = S_r[S_r[r] + S_{r-1}[r]] = S_r[0 + r] = S_r[r] = 0$. Also, when $j_r = r + r$, since $S_r[r] = r$ and $Z_r = S_r[S_r[r] + S_{r-1}[r]] = S_r[r + r] \neq r$, the probability of $Z_r = r$ is zero. Thus, the conditions of $j_r \neq r, r + r$ are necessary for $Z_r = r$. Then, it is estimated as

$$\Pr(S_{r-1}[r] = r \wedge S_r[r] = j_r - r \wedge j_r \neq r, r + r) = p_{r-1,r} \cdot \frac{1}{N} \cdot \frac{N-2}{N}.$$

Case 3 : $S_{r-1}[r] \neq 0 \wedge S_r[r] = r - S_{r-1}[r]$
The equation of $Z_r = S_r[r - S_{r-1}[r] + S_{r-1}[r]] = S_r[r]$ holds. Then, $S_r[r] = r - S_{r-1}[r]$ is not r, because $S_{r-1}[r]$ is not 0. Thus, it is estimated as

$$\Pr(S_{r-1}[r] \neq 0 \wedge S_r[r] = r - S_{r-1}[r]) = (1 - p_{r-1,0}) \cdot \frac{1}{N}.$$

Case 4 : $S_{r-1}[r] \neq 0 \wedge S_r[r] = r$

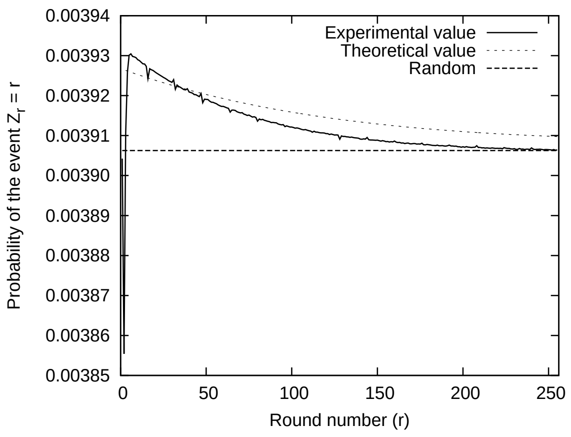


Figure 9.5: Theoretical values and experimental values of $Z_r = r$

The output is expressed as $Z_r = S_r[r + S_{r-1}[r]]$. According to the equation of $S_{r-1}[r] \neq 0$, The probability of $Z_r = r$ is zero. Thus, it is estimated as

$$\Pr(S_{r-1}[r] \neq (0, r) \wedge S_r[r] = r - S_{r-1}[r]) = (1 - p_{r-1,0}) \cdot \frac{1}{N}.$$

Assuming that in other cases, $Z_r = r$ holds with probability of $1/N$, the probability of $\Pr(Z_r = r)$ is estimated as

$$\begin{aligned} \Pr(Z_r = r) \quad \approx \quad & p_{r-1,0} \cdot \frac{1}{N} + p_{r-1,r} \cdot \frac{1}{N} \cdot \frac{N-2}{N} + \\ & (1 - p_{r-1,0} \cdot \frac{1}{N} - p_{r-1,r} \cdot \frac{1}{N} - (1 - p_{r-1,0}) \cdot \frac{1}{N} \cdot 2) \cdot \frac{1}{N}. \end{aligned}$$

$\square$ $\square$

Here, $p_{r-1,r}$ and $p_{r-1,0}$ are obtained from Theorem 6. Figure 9.5 shows the comparison of theoretical values and experimental values of $Z_r = r$ for 2^{40} randomly-chosen keys when $N = 256$. Since the theoretical values do not exactly coincide with the experimental values, we do not claim that Theorem 10 completely prove this bias. We guess that several minor events are not covered in our approach. However, the order of the bias seems to be well matched. At least it can be said that the main event causing this bias is discovered.

9.3.4 Extended Keylength-Dependent Biases

Extended keylength-dependent biases, which are extensions of keylength-dependent biases [133, 36], are the bias of $Z_\ell = -\ell$ when the key length is ℓ bytes. For example, when using a 128-bit key (16 bytes), Z_{16} is biased to -16 ($= 240$). In addition to it, we show that when the key length is ℓ bytes, $Z_{x\cdot\ell}$ is also biased to $-x \cdot \ell$ ($x = 2, 3, 4, 5, 6, 7$), e.g., $Z_r = -r$ for $r = 32, 48, 64, 80, 96, 112$, assuming $\ell = 16$. Importantly, the extended keylength-dependent biases are much stronger than the other known biases such as $Z_r = 0$ and $Z_r = r$. Table 9.1 shows experimental values of the extended keylength-dependent bias $Z_r = -r$, $Z_r = 0$, and $Z_r = r$ for 2^{40} randomly-chosen keys, when r is a multiple of the key length, $\ell = 16$ in this case.

Table 9.1: Experimental values of $Z_r = -r$, $Z_r = 0$ and $Z_r = r$

r	$\Pr(Z_r = -r)$	$\Pr(Z_r = 0)$	$\Pr(Z_r = r)$
16	$2^{-8} \cdot (1 + 2^{-4.811})$	$2^{-8} \cdot (1 + 2^{-7.714})$	$2^{-8} \cdot (1 + 2^{-7.762})$
32	$2^{-8} \cdot (1 + 2^{-5.383})$	$2^{-8} \cdot (1 + 2^{-7.880})$	$2^{-8} \cdot (1 + 2^{-7.991})$
48	$2^{-8} \cdot (1 + 2^{-5.938})$	$2^{-8} \cdot (1 + 2^{-8.043})$	$2^{-8} \cdot (1 + 2^{-8.350})$
64	$2^{-8} \cdot (1 + 2^{-6.496})$	$2^{-8} \cdot (1 + 2^{-8.244})$	$2^{-8} \cdot (1 + 2^{-8.664})$
80	$2^{-8} \cdot (1 + 2^{-7.224})$	$2^{-8} \cdot (1 + 2^{-8.407})$	$2^{-8} \cdot (1 + 2^{-9.052})$
96	$2^{-8} \cdot (1 + 2^{-7.911})$	$2^{-8} \cdot (1 + 2^{-8.577})$	$2^{-8} \cdot (1 + 2^{-9.351})$
112	$2^{-8} \cdot (1 + 2^{-8.666})$	$2^{-8} \cdot (1 + 2^{-8.747})$	$2^{-8} \cdot (1 + 2^{-9.732})$

The probability of these biases is given as Theorem 11.

Theorem 11. *When $r = x \cdot \ell$ $(x = 1, 2, \ldots, 7)$, the probability of $\Pr(Z_r = -r)$ is approximately*

$$\Pr(Z_r = -r) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right) \cdot \gamma_r + (1 - \delta_r) \cdot \frac{1}{N},$$

where

$$\begin{aligned}\gamma_r &= \frac{1}{N^2} \cdot \left(1 - \frac{r+1}{N}\right) \\ &\quad \cdot \sum_{y=r+1}^{N-1} \left(1 - \frac{1}{N}\right)^{y} \cdot \left(1 - \frac{2}{N}\right)^{y-r} \cdot \left(1 - \frac{3}{N}\right)^{N-y+2r-4},\end{aligned}$$

and $\delta_r = \Pr(S_r[j_r] = 0) = \Pr(S_{r-1}[r] = 0)$.

Figure 9.9 shows our experimental values for 2^{40} randomly-chosen keys and theoretical values of these extended keylength-dependent biases. Since theoretical and experimental values have almost the same value, theoretical values are correctly approximated.

Proof of Theorem 11

In order to prove Theorem 11, we give following Lemma 2 and Theorem 12, which are extensions of Lemma 2 and Theorem 3 in [35]. Let (S^K_r, i^K_r, j^K_r) be (S, i, j) of the r-th round in the KSA, respectively.

Lemma 2. *When $r = x \cdot \ell$ $(x = 1, 2, \ldots, 7)$, the probability of $\Pr(S^K_{r+1}[r-1] = -r \wedge S^K_{r+1}[r] = 0)$ is approximately*

$$\Pr(S^K_{r+1}[r-1] = -r \wedge S^K_{r+1}[r] = 0) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right) \cdot \alpha_r,$$

where $\alpha_r = \frac{1}{N} \cdot \left(1 - \frac{3}{N}\right)^{r-2} \cdot \left(1 - \frac{r+1}{N}\right)$.

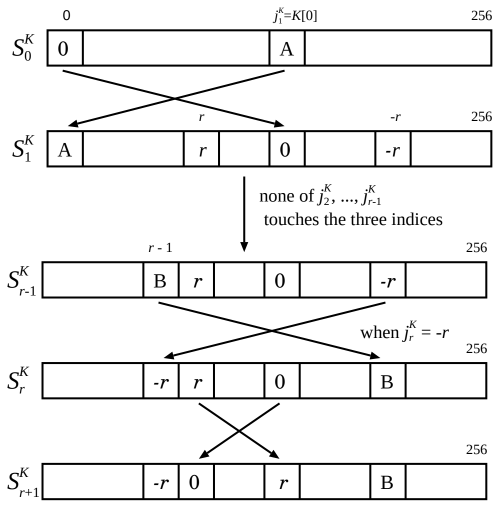


Figure 9.6: Event for bias of $S^K_{r+1}[r-1] = -r \wedge S^K_{r+1}[r] = 0$

Proof. The event of $(S^K_{r+1}[r-1] = -r \wedge S^K_{r+1}[r] = 0)$ consists of following events. In the first round of the KSA, when $i^K_1 = 0$ and $j^K_1 = K[0]$, the value 0 is swapped for the value of $S^K_0[K[0]]$ with probability of one. The index j^K_1 requires $j^K_1 = K[0] \notin \{r-1, r, -r\}$, so that the values $r-1$, r, $-r$ are not swapped in the first round of the KSA, respectively. In addition to it, it is required that $K[0] \notin \{1, 2, \ldots, r-2\}$, so that the value 0 at index $K[0]$ is not touched by these values of i^K during the next $r-2$ rounds of the KSA. This happens with probability of $\left(1 - \frac{r+1}{N}\right)$. From round 2 to $r-1$ of the KSA, $j^K_2, j^K_3, \ldots, j^K_{r-1}$ do not touch the three indices $\{r, -r, K[0]\}$, respectively. This happens with probability of $\left(1 - \frac{3}{N}\right)^{r-2}$. In the r-th round of the KSA, if the index j^K_r has the index $-r$, which happens with probability of $1/N$, the value $-r$ is swapped into the index $r-1$. In the $(r+1)$-th round of the KSA, when $i^K_{r+1} = r$ and $j^K_{r+1} = j^K_r + S^K_r[r] + K[r] = -r + r + K[0] = K[0]$, the value $S^K_r[r]$ is swapped for the value $S^K_r[K[0]]$, and from the above discussion, this index contains the value 0. Considering the above events to be independent, the probability that all of above events happen together is given by $\alpha_r = \frac{1}{N} \cdot \left(1 - \frac{3}{N}\right)^{r-2} \cdot \left(1 - \frac{r+1}{N}\right)$. Assuming that in other cases, $(S^K_{r+1}[r-1] = -r \wedge S^K_{r+1}[r] = 0)$ holds with probability of $1/N^2$, the probability of $\Pr(S^K_{r+1}[r-1] = -r \wedge S^K_{r+1}[r] = 0)$ is estimated as

$$\Pr(S^K_{r+1}[r-1] = -r \wedge S^K_{r+1}[r] = 0) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right) \cdot \alpha_r.$$

□

Figure 9.6 shows the major path of $S^K_{r+1}[r-1] = -r \wedge S^K_{r+1}[r] = 0$.

Theorem 12. *When* $r = x \cdot \ell$ $(x = 1, 2, \ldots, 7)$, *the probability of* $\Pr(Z_r = -r \wedge S_r[j_r] = 0)$ *is approximately*

$$\Pr(Z_r = -r \wedge S_r[j_r] = 0) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right) \cdot \gamma_r,$$

where

$$\begin{aligned}\gamma_r &= \frac{1}{N^2}\cdot\left(1-\frac{r+1}{N}\right)\\ &\cdot\sum_{y=r+1}^{N-1}\left(1-\frac{1}{N}\right)^{y}\cdot\left(1-\frac{2}{N}\right)^{y-r}\cdot\left(1-\frac{3}{N}\right)^{N-y+2r-4}.\end{aligned}$$

Proof. From the algorithm of the PRGA, we have $j_r = j_{r-1} + S_{r-1}[r]$. Hence, $S_r[j_r] = S_{r-1}[r] = 0$ implies $j_r = j_{r-1}$. In this case, an output Z_r is expressed as

$$Z_r = S_r[S_r[i_r] + S_r[j_r]] = S_r[S_{r-2}[r-1]].$$

Then, let us consider $\Pr(S_r[S_{r-2}[r-1]] = -r \wedge S_r[j_r] = 0)$.

The major path for the joint event $(S^K_{r+1}[r-1] = -r \wedge S^K_{r+1}[r] = 0)$ constitutes the first part of our main path leading to the target event. The second part can be constructed as follows. In an index $y \in [r+1, N-1]$, if the j^K do not touch the index y, we have $S^K_y[y] = y$ with probability of $\left(1-\frac{1}{N}\right)^y$. From round $r+2$ to y of the KSA, j^K do not touch the two indices $\{r-1, r\}$, respectively. This happens with probability of $\left(1-\frac{2}{N}\right)^{y-r-1}$. In the $(y+1)$-th round of the KSA, if the index j^K_{y+1} has the index $r-1$, which happens with probability of $1/N$, the value y is swapped for the value $-r$. Then, the value $-r$ moves to $S^K_{y+1}[y] = S^K_{y+1}[S^K_{y+1}[r-1]]$. For the remaining $N-y-1$ rounds of the KSA and for the first $r-1$ rounds of the PRGA, the j^K or j values should not touch the indices $\{r-1, S[r-1], r\}$, respectively. This happens with probability of $\left(1-\frac{3}{N}\right)^{N-y+r-2}$. Now, we have $(S_{r-1}[S_{r-2}[r-1]] = -r \wedge S_{r-1}[r] = 0)$. And then, we should also have $j_r \notin \{r-1, y\}$ for $S_r[S_{r-2}[r-1]] = -r$. The probability of this condition is $\left(1-\frac{2}{N}\right)$. Then, from algorithm of the PRGA, the output is $Z_r = S_r[S_{r-2}[r-1]] = -r$. Considering the above events to be independent, the probability that the second part events happen together is given by

$$\alpha'_r = \frac{1}{N}\cdot\sum_{y=r+1}^{N-1}\left(1-\frac{1}{N}\right)^{y}\cdot\left(1-\frac{2}{N}\right)^{y-r}\cdot\left(1-\frac{3}{N}\right)^{N-y+r-2}.$$

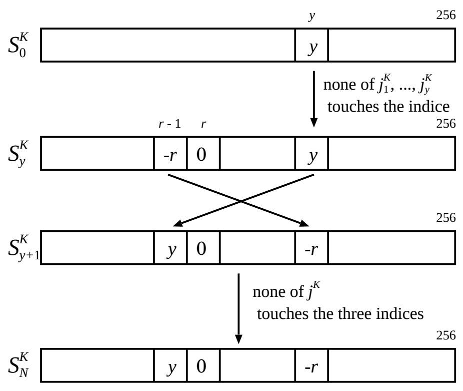
Then, the probability that all of the events happen together is estimated as

$$\begin{aligned}\gamma_r &= \alpha_r\cdot\alpha'_r\\ &= \frac{1}{N^2}\cdot\left(1-\frac{r+1}{N}\right)\\ &\cdot\sum_{y=r+1}^{N-1}\left(1-\frac{1}{N}\right)^{y}\cdot\left(1-\frac{2}{N}\right)^{y-r}\cdot\left(1-\frac{3}{N}\right)^{N-y+2r-4}.\end{aligned}$$

Assuming that in other cases, $Z_r = -r \wedge S_r[j_r] = 0$ holds with probability of $1/N^2$, the probability of $\Pr(Z_r = -r \wedge S_r[j_r] = 0)$ is approximately

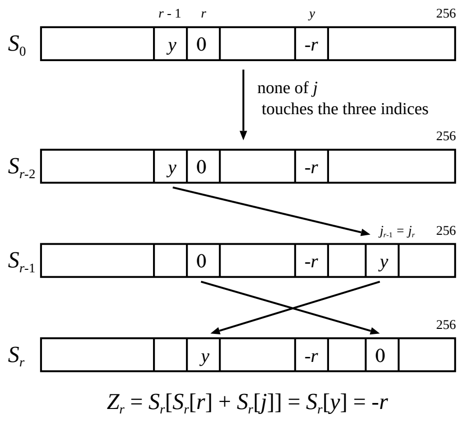
$$\Pr(Z_r = -r \wedge S_r[j_r] = 0) \approx \frac{1}{N^2} + \left(1-\frac{1}{N^2}\right)\cdot\gamma_r.$$

□



S_0^K
S_y^K
S_{y+1}^K
S_N^K
y
256
none of $j_1^K, ..., j_y^K$ touches the indice
r - 1
r
-r
0
none of j^K touches the three indices

Figure 9.7: Event for bias of $Z_r = -r \wedge S_r[j_r] = 0$ on KSA



S_0
S_{r-2}
S_{r-1}
S_r
r - 1
r
y
256
-r
0
none of j touches the three indices
$j_{r-1} = j_r$
$Z_r = S_r[S_r[r] + S_r[j]] = S_r[y] = -r$

Figure 9.8: Event for bias of $Z_r = -r \wedge S_r[j_r] = 0$ on PRGA

Figure 9.7 and 9.8 show the major path of $Z_r = -r \wedge S_r[j_r] = 0$.

Using these extended joint events, the theorem 11 is proved as follows.

Proof. We can write $\Pr(Z_r = -r) = \Pr(Z_r = -r \wedge S_r[j_r] = 0) + \Pr(Z_r = -r \wedge S_r[j_r] \neq 0)$, where the first term is given by Theorem 12. When $S_r[j_r] \neq 0$, the event $Z_r = -r$ can be assumed to hold with probability of $1/N$. Then, the probability of $\Pr(Z_r = -r)$ is estimated as

$$\Pr(Z_r = -r) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right) \cdot \gamma_r + (1 - \delta_r) \cdot \frac{1}{N}.$$

□

9.3.5 Cumulative Bias Set of First 257 Bytes

When $N = 256$, a set of strong biases in $Z_1, Z_2, \ldots, Z_{255}$ is given in Table 9.2. Our new biases, namely the ones involving Z_1, Z_3, Z_{32}, Z_{48}, Z_{64}, Z_{80}, Z_{96}, Z_{112}, are included. Here, let us compare between the biases of $Z_r = 0$ [34, 35] and $Z_r = r$, whose probabilities are of the same order, and are very close in the range

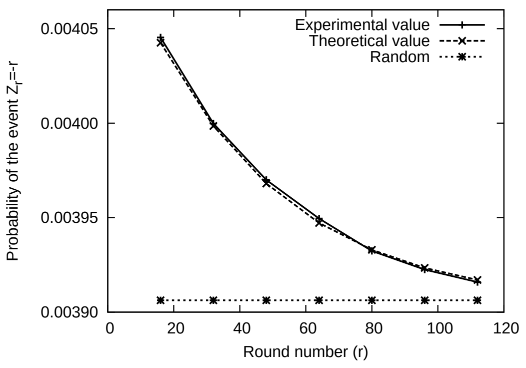


Figure 9.9: Experimental values and theoretical values of $Z_r = -r$ when $\ell = 16$ for $r = 16, 32, 48, 64, 80, 96, 112$

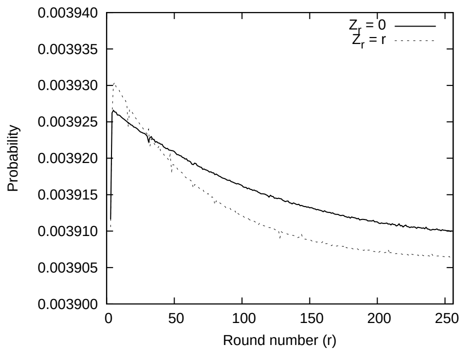


Figure 9.10: Comparison between $Z_r = 0$ and $Z_r = r$ for $3 \leq r \leq 255$

$3 \leq r \leq 255$. According to our experiments with 2^{40} randomly-chosen keys (see Fig. 9.10), $Z_r = r$ is stronger than $Z_r = 0$ in $Z_5, Z_6, \ldots, Z_{31}$. Thus we choose the bias $Z_r = r$ in $Z_5, Z_6, \ldots, Z_{31}$ and the bias $Z_r = 0$ in the other cases as the strongest bias except for the cases involving Z_3, Z_{16}, Z_{32}, Z_{48}, Z_{64}, Z_{80}, Z_{96}, Z_{112}. Besides, we experimentally found two new biases for the events $Z_{256} \neq 0$ and $Z_{257} = 0$, and added these to our bias set, while we could not provide the theoretical proofs. Note that it is experimentally confirmed that biases of $Z_2, Z_3, \ldots, Z_{257}$ included in our bias set are strongest known biases amongst all the positive and negative biases that have been discovered for these bytes.

For the first time, we propose a cumulative list of strongest known biases in the initial bytes of RC4 that can be exploited in a practical attack against the broadcast mode of the cipher.

9.4 Experimental Results of Plaintext Recovery Attack

We demonstrate a plaintext recovery attack using our cumulative bias set of first 257 bytes by a computer experiment, when $N = 256$, and estimate the number of required ciphertexts and the probability of success for our attack. The details of our

Table 9.2: Cumulative bias set of first 257 bytes

r	Strongest known bias of Z_r	Prob.(Theoretical)	Prob.(Experimental)
1	$Z_1 = 0 \| Z_2 = 0$ (Our)	$2^{-8} \cdot (1 + 2^{-1.009})$	$2^{-8} \cdot (1 + 2^{-1.036})$
2	$Z_2 = 0$ [33]	$2^{-8} \cdot (1 + 2^{0})$	$2^{-8} \cdot (1 + 2^{0.002})$
3	$Z_3 = 131$ (Our)	$2^{-8} \cdot (1 + 2^{-8.089})$	$2^{-8} \cdot (1 + 2^{-8.109})$
4	$Z_4 = 0$ [34]	$2^{-8} \cdot (1 + 2^{-7.581})$	$2^{-8} \cdot (1 + 2^{-7.611})$
5–15	$Z_r = r$ (Our)	max: $2^{-8} \cdot (1 + 2^{-7.627})$ min: $2^{-8} \cdot (1 + 2^{-7.737})$	max: $2^{-8} \cdot (1 + 2^{-7.335})$ min: $2^{-8} \cdot (1 + 2^{-7.535})$
16	$Z_{16} = 240$ [36]	$2^{-8} \cdot (1 + 2^{-4.841})$	$2^{-8} \cdot (1 + 2^{-4.811})$
17–31	$Z_r = r$ (Our)	max: $2^{-8} \cdot (1 + 2^{-7.759})$ min: $2^{-8} \cdot (1 + 2^{-7.912})$	max: $2^{-8} \cdot (1 + 2^{-7.576})$ min: $2^{-8} \cdot (1 + 2^{-7.839})$
32	$Z_{32} = 224$ (Our)	$2^{-8} \cdot (1 + 2^{-5.404})$	$2^{-8} \cdot (1 + 2^{-5.383})$
33–47	$Z_r = 0$ [34]	max: $2^{-8} \cdot (1 + 2^{-7.897})$ min: $2^{-8} \cdot (1 + 2^{-8.050})$	max: $2^{-8} \cdot (1 + 2^{-7.868})$ min: $2^{-8} \cdot (1 + 2^{-8.039})$
48	$Z_{48} = 208$ (Our)	$2^{-8} \cdot (1 + 2^{-5.981})$	$2^{-8} \cdot (1 + 2^{-5.938})$
49–63	$Z_r = 0$ [34]	max: $2^{-8} \cdot (1 + 2^{-8.072})$ min: $2^{-8} \cdot (1 + 2^{-8.224})$	max: $2^{-8} \cdot (1 + 2^{-8.046})$ min: $2^{-8} \cdot (1 + 2^{-8.238})$
64	$Z_{64} = 192$ (Our)	$2^{-8} \cdot (1 + 2^{-6.576})$	$2^{-8} \cdot (1 + 2^{-6.496})$
65–79	$Z_r = 0$ [34]	max: $2^{-8} \cdot (1 + 2^{-8.246})$ min: $2^{-8} \cdot (1 + 2^{-8.398})$	max: $2^{-8} \cdot (1 + 2^{-8.223})$ min: $2^{-8} \cdot (1 + 2^{-8.376})$
80	$Z_{80} = 176$ (Our)	$2^{-8} \cdot (1 + 2^{-7.192})$	$2^{-8} \cdot (1 + 2^{-7.224})$
81–95	$Z_r = 0$ [34]	max: $2^{-8} \cdot (1 + 2^{-8.420})$ min: $2^{-8} \cdot (1 + 2^{-8.571})$	max: $2^{-8} \cdot (1 + 2^{-8.398})$ min: $2^{-8} \cdot (1 + 2^{-8.565})$
96	$Z_{96} = 160$ (Our)	$2^{-8} \cdot (1 + 2^{-7.831})$	$2^{-8} \cdot (1 + 2^{-7.911})$
97–111	$Z_r = 0$ [34]	max: $2^{-8} \cdot (1 + 2^{-8.592})$ min: $2^{-8} \cdot (1 + 2^{-8.741})$	max: $2^{-8} \cdot (1 + 2^{-8.570})$ min: $2^{-8} \cdot (1 + 2^{-8.722})$
112	$Z_{112} = 144$ (Our)	$2^{-8} \cdot (1 + 2^{-8.500})$	$2^{-8} \cdot (1 + 2^{-8.666})$
113–255	$Z_r = 0$ [34]	max: $2^{-8} \cdot (1 + 2^{-8.763})$ min: $2^{-8} \cdot (1 + 2^{-10.052})$	max: $2^{-8} \cdot (1 + 2^{-8.760})$ min: $2^{-8} \cdot (1 + 2^{-10.041})$
256	$Z_{256} = 0$ (negative bias) (Our)	N/A	$2^{-8} \cdot (1 - 2^{-9.407})$
257	$Z_{257} = 0$ (Our)	N/A	$2^{-8} \cdot (1 + 2^{-9.531})$

experiment are as follows.

Step 1 Randomly generate a target plaintext P.

Step 2 Encrypt P with 2^x randomly-chosen keys, and obtain 2^x ciphertexts C.

Step 3 Find most frequent byte in each byte, and extract P_r, assuming $P_r = C_r \oplus Z_r$ where Z_r is the value of the keystream byte from our bias set.

In the case of P_1, the method mentioned in Section 8.3 is used for efficient extraction of P_1. Specifically, after P_2 is recovered, we extract P_1 by using the conditional bias such that $Z_1 = 0$ when $Z_2 = 0$.

We perform the above experiment for 256 different plaintexts in the cases where $2^6, 2^7, \ldots, 2^{35}$ ciphertexts with randomly-chosen keys are given. Figure 9.11 shows the probability of successfully recovering the values of P_1, P_2, P_3, P_5, and P_{16} for each amount of ciphertexts. Here, the success probability is estimated by the number of correctly-extracted plaintexts for each byte. For example, if the target byte of only 100 plaintexts out of 256 plaintexts can be correctly recovered, the probability is estimated as 0.39 (= 100/256). The second byte of plaintext P_2 can be extracted from 2^{12} ciphertexts with probability one. In previous attacks such as

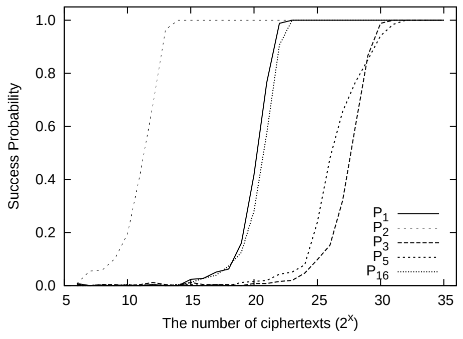


Figure 9.11: Relation of the number of ciphertexts and success probability of recovering P_1, P_2, P_3, P_5, and P_{16}

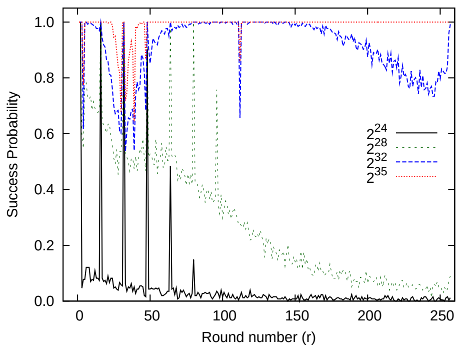


Figure 9.12: Success probability of extracting P_r $(1 \leq r \leq 257)$ with different number of samples (one candidate)

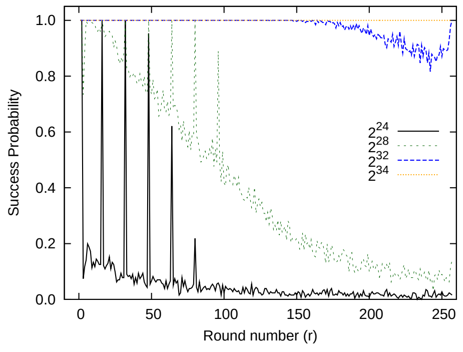


Figure 9.13: Success probability of extracting P_r $(1 \leq r \leq 257)$ with different number of samples (two candidates)

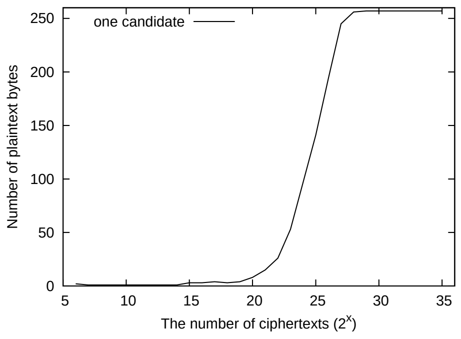


Figure 9.14: The number of plaintext bytes that are extracted with five times higher than that of a random guess

the MS attack [33] and the MPS attack [34], the number of required ciphertexts is theoretically estimated only in terms of the lower bound Ω. Our results first reveal the concrete number of ciphertexts, and the corresponding success probability.

Figure 9.12 shows that the success probability of extracting each byte P_r $(1 \leq r \leq 257)$ when 2^{24}, 2^{28}, 2^{32}, 2^{35} ciphertexts are given. Note that the probability of a random guess is $1/256 = 0.00390625$. Given 2^{32} ciphertexts, all bytes of $P_1, P_2, \ldots, P_{257}$ can be extracted with probability more than 0.5. In addition, most bytes can be extracted with probability more than 0.8. Also, the bytes having stronger bias such as P_1, P_2, P_{16}, P_{32}, P_{48}, P_{64}, are extracted from only 2^{24} ciphertexts with high probability. However, even if 2^{35} ciphertexts are given, the probability does not become one in some bytes. It is guessed that in such bytes, the difference of probability of the strongest known bias (as in our cumulative bias set) and the second one is very small. Thus, more ciphertexts are required for an attack with probability one.

We additionally utilize the second most frequent byte in the ciphertexts for extracting plaintext bytes. In other words, two candidates are obtained by using the relation of $P_r = C_r \oplus Z_r$, where C_r are most and second most frequent ciphertext bytes and Z_r is chosen from our bias set. This result is shown in Fig. 9.13, and its success probability is estimated as the probability that the guess for the correct plaintext byte is narrowed down to two possible candidates. Note that the probability of a random guess for such a scenario is $2/256 = 0.0078125$. Given 2^{34} ciphertexts, each byte of $P_1, P_2, \ldots, P_{257}$ can be extracted with probability one. In this case, although we can not obtain the correct byte of the plaintext, it is narrowed down to only two candidates. For the experiments of Fig. 9.12 and 9.13, it requires about one day if one uses a single CPU core (Intel(R) Core(TM) i7 CPU 920@ 2.67GHz) to obtain the result of one plaintext, where 256 plaintexts are used.

Figure 9.14 shows the number of plaintext bytes that are extracted with five times higher probability than that of a random guess, i.e., where the success probability is more than $\frac{5}{256}$. Given 2^{29} ciphertexts, all the plaintext bytes $P_1, P_2, \ldots, P_{257}$ are guessed with much higher probability than random guesses.

9.5 How to Recover Bytes of the Plaintext after P_{258}

In this section, we propose an efficient method to recover later bytes of the plaintext, namely bytes after P_{258}. The method using our bias in initial bytes is not directly applied to extract these bytes, because it exploits biases existing in only the initial keystream. For the extraction of the later bytes, a long-term bias, which occurs in any keystream bytes, is utilized. In particular, the digraph repetition bias (also called $ABSAB$ bias) proposed by Mantin [37], which is the strongest known long-term bias, is used. Combining it with our cumulative bias set of $Z_1, Z_2, \ldots, Z_{257}$, we can sequentially recover bytes of a plaintext, even after P_{258}, given only the ciphertexts.

9.5.1 Best Known Long-term Bias ($ABSAB$ bias)

$ABSAB$ bias is statistical biases of the digraph distribution in the RC4 keystream [37]. Specifically, digraphs AB tend to repeat with short gaps S between them, e.g., $ABAB$, $ABCAB$ and $ABCDAB$, where gap S is defined as zero, C, and CD, respectively. The detail of $ABSAB$ bias is expressed as follows,

$$Z_r \mid\mid Z_{r+1} = Z_{r+2+G} \mid\mid Z_{r+3+G} \quad \text{for } G \geq 0, \tag{9.1}$$

where $||$ is a concatenation. The probability that Eq. (9.1) holds is given as Theorem 13.

Theorem 13. [37] *For small values of G the probability of the pattern $ABSAB$ in RC4 keystream, where S is a G-byte string, is $(1 + e^{(-4-8G)/N}/N) \cdot 1/N^2$.*

For the enhancement of these biases, combining use of $ABSAB$ biases with different G is considered by using the following lemma for the discrimination.

Lemma 3. [37] *Let X and Y be two distributions and suppose that the independent events $\{E_i\colon\ 1 \leq i \leq k\ \}$ occur with probabilities $p_X(E_i) = p_i$ in X and $p_Y(E_i) = (1 + b_i) \cdot p_i$ in Y. Then the discrimination D of the distributions is $\sum_i p_i \cdot b_i^2$.*

The number of required samples for distinguishing the biased distribution from the random distribution with probability of $1 - \alpha$ is given as the following lemma.

Lemma 4. [37] *The number of samples that is required for distinguishing two distributions that have discrimination D with success rate $1-\alpha$ (for both directions) is $(1/D)\cdot(1-2\alpha)\cdot log_2\frac{1-\alpha}{\alpha}$.*

This lemma shows that in the broadcast RC4 attack, given D and the number of samples $N_{ciphertext}$, the success probability for distinguishing the distribution of correct candidate plaintext byte (the biased distribution) from the distribution of one wrong candidate of plaintext byte (a random distribution) is a constant. $\Pr_{distingush}$ denotes this probability.

9.5.2 Plaintext Recovery Method Using *ABSAB* Bias and Our Bias Set

The following equation allows us to efficiently use *ABSAB* bias in the broadcast RC4 attack.

$$\begin{aligned}&(C_r \mid\mid C_{r+1}) \oplus (C_{r+2+G} \mid\mid C_{r+3+G})\\&= (P_r \oplus Z_r \mid\mid P_{r+1} \oplus Z_{r+1}) \oplus (P_{r+2+G} \oplus Z_{r+2+G} \mid\mid P_{r+3+G} \oplus Z_{r+3+G})\\&= (P_r \oplus P_{r+2+G} \oplus Z_r \oplus Z_{r+2+G} \mid\mid P_{r+1} \oplus P_{r+3+G} \oplus Z_{r+1} \oplus Z_{r+3+G}). \quad (9.2)\end{aligned}$$

Assuming that Eq. (9.1) (the event of the *ABSAB* bias) holds, the relation of plaintexts and ciphertexts without keystreams is obtained, i.e., $(C_r \mid\mid C_{r+1}) \oplus (C_{r+2+G} \mid\mid C_{r+3+G}) = (P_r \oplus P_{r+2+G} \mid\mid P_{r+1} \oplus P_{r+3+G}) = (P_r \mid\mid P_{r+1}) \oplus (P_{r+2+G} \mid\mid P_{r+3+G})$.

However, in the straight way, we can not combine these relations with different G to enhance the biases, as we do in the distinguishing attack setting. When the value of G is different, the above equation is surely different even if r is properly chosen. For example, in the cases of (r and $G = 1$) and ($r + 1$ and $G = 0$), right parts of equations are given as $(P_r \mid\mid P_{r+1}) \oplus (P_{r+3} \mid\mid P_{r+4})$ and $(P_{r+1} \mid\mid P_{r+2}) \oplus (P_{r+3} \mid\mid P_{r+4})$, respectively. Thus, due to independent use of these equations with different G, we are not able to efficiently make use of *ABSAB* bias in the broadcast setting.

In order to get rid of this problem, we give a method that sequentially recovers the plaintext after P_{258} with the knowledge of pre-guessed plaintext bytes. For example, in the cases of (r and $G = 1$) and ($r + 1$ and $G = 0$), if P_r, P_{r+1}, and P_{r+2} are already known, the two equations with respected to $(P_{r+3} \mid\mid P_{r+4})$ is obtained by transposing P_r, P_{r+1}, and P_{r+2} to the left part of the equation. Then, these equations with different G can be merged.

Suppose that $P_1, P_2, \dots, P_{257}$ are guessed by our cumulative bias set of the initial bytes, where the success probability of finding these bytes are evaluated in Section 8.4. Then we aim to sequentially find P_r for $r = 258, 259, \dots, P_{MAX}$ by using *ABSAB* biases of $G = 0, 1, \dots, G_{MAX}$. The detailed procedures are given as follows.

Step 1 Obtain $C_{258-3-G_{MAX}}, C_{258-2-G_{MAX}}, \dots, C_{P_{MAX}}$ in each ciphertext, and make frequency tables $T_{count}[r][G]$ of $(C_{r-3-G} \mid\mid C_{r-2-G}) \oplus (C_{r-1} \mid\mid C_r)$ for all $r = 258, 259, \dots, P_{MAX}$ and $G = 0, 1, \dots, G_{MAX}$, where $(C_{r-3-G} \mid\mid C_{r-2-G}) \oplus (C_{r-1} \mid\mid C_r) = (P_{r-3-G} \mid\mid P_{r-2-G}) \oplus (P_{r-1} \mid\mid P_r)$ only if Eq. (9.1) holds.

Step 2 Set $r = 258$.

Step 3 Guess the value of P_r.

Step 3.1 For $G = 0, 1, \ldots, G_{MAX}$, convert $T_{count}[r][G]$ into a frequency table $T_{marge}[r]$ of $(P_{r-1} \mid\mid P_r)$ by using pre-guessed values of $P_{r-3-G_{MAX}}, \ldots, P_{r-2}$, and merge counter values of all tables.

Step 3.2 Make a frequency table $T_{guess}[r]$ indexed by only P_r from $T_{marge}[r]$ with knowledge of the P_{r-1}. To put it more precisely, using a pre-guessed value of P_{r-1}, only Tables $T_{marge}[r]$ corresponding to the value of P_{r-1} is taken into consideration. Finally, regard most frequency one in table $T_{guess}[r]$ as the correct P_r.

Step 4 Increment r. If $r = P_{MAX} + 1$, terminate this algorithm. Otherwise, go to Step 3.

The bytes of the plaintext are correctly extracted from $T_{marge}[r]$ only if it is distinguished from other $N^2 - 1$ wrong candidate distributions. Assuming that wrong candidates are randomly distributed, a probability of the correct extraction from $T_{marge}[r]$ is estimated as $(\Pr_{distingush})^{N^2-1}$. In Step 3.2, our method converts $T_{marge}[r]$ into $T_{guess}[r]$ by using knowledge of P_{r-1}, where $T_{guess}[r]$ has $N-1$ wrong candidates. It enables us to reduce the number of wrong candidates from $N^2 - 1$ to $N - 1$. Then, a probability of the correct extraction from $T_{guess}[r]$ is estimated as $(\Pr_{distingush})^{N-1}$, which is $1/(\Pr_{distingush})^{N+1}$ times higher than that of $T_{marge}[r]$. Therefore, the table reduction technique of Step 3.2 enables us to further optimize the attack.

Experimental Results:

We perform practical experiments using our algorithm to find P_{258}, P_{259}, P_{260}, and P_{261} ($P_{MAX} = 261$). As a parameter of $ABSAB$ bias, $G_{MAX} = 63$ is chosen, because the increase of D is converged around $G_{MAX} = 63$. Then, D is estimated as $D = 2^{-28.0}$. The success probability of our algorithm for recovering P_r ($r \geq 258$) when 2^{30} to 2^{34} ciphertexts are given is shown in Table 9.3, where the number of tests is 256. Note that $P_1, P_2, \ldots, P_{257}$ are obtained by using our bias set (candidate one) with success probability as shown in Fig. 9.12. For this experiment, it requires about one week if one uses a single CPU core (Intel(R) Core(TM) i7 CPU 920@ 2.67GHz) to get the result of one plaintext, where 256 plaintexts are used.

Interestingly, given 2^{34} ciphertexts, P_{258}, P_{259}, P_{260}, and P_{261} can be recovered with probability one, while the success probability of some bytes in $P_1, P_2, \ldots, P_{257}$ is not one. Combining multiple biases allows us to omit negative effects of some uncorrected value of $P_1, P_2, \ldots, P_{257}$. Although our experiment is performed until P_{261}, the success probability is expected not to change even in the case of later bytes, because $ABSAB$ bias is a long-term bias.

Let us discuss the success probability of extracting bytes after P_{262} when 2^{34} ciphertexts are given. According to Lemma 4 and $D = 2^{-28.0}$, 2^{34} ciphertexts allow us to distinguish an RC4 keystream from a random stream with the probability of $\Pr_{distinguish} = 1 - 10^{-19}$. Then, assuming that wrong candidates are randomly distributed, the probability of correctly extracting the candidate from $(N-1)$ wrong candidates is estimated as $(\Pr_{distinguish})^{N-1}$. Therefore, our method enables to extract consecutive $(257 + X)$ bytes of a plaintext with the probability of $((\Pr_{distinguish})^{N-1})^X = (\Pr_{distinguish})^{(N-1)\cdot X}$. For instance, when $X = 2^{40}$ and $X = 2^{50}$, the success probabilities are estimated as 0.99997 and 0.97170, respectively.

Table 9.3: Success probability of our algorithm for recovering P_r ($r \geq 258$).

# of ciphertexts	P_{258}	P_{259}	P_{260}	P_{261}
2^{30}	0.003906	0.003906	0.000000	0.000000
2^{31}	0.039062	0.007812	0.003906	0.007812
2^{32}	0.386719	0.152344	0.070312	0.027344
2^{33}	0.964844	0.941406	0.921875	0.902344
2^{34}	1.000000	1.000000	1.000000	1.000000

As a result, by using our sequential method, a large amount of plaintext bytes, e.g., first 2^{50} bytes $\approx$ 1000 T bytes, is recovered from 2^{34} ciphertext with a probability of almost one. Therefore, it can be said that our attack is a full plaintext recovery attack on broadcast RC4, the first of its kind proposed in the literature.

9.6 Key (State) Recovery Attack

Our new biases are also applicable to key (state) recovery attacks by using the relation among the keystream, the state and the key. The conditional bias, $Z_3 = 131$ bias and extended keylength-dependent biases, whose occurrence probabilities are completely proved, can be used for (state) key recovery attacks.

Interestingly, this attack is feasible even if only ciphertexts are given in the broadcast setting. In particular, the aforementioned plaintext recovery attack allow us to extract target bytes of the plaintext from only ciphertexts. Then, we observe target bytes of the keystream, and then mount key (state) guess attacks if target bytes are expected values.

9.6.1 Key (State) Recovery Attack Using Conditional Bias

When $Z_1 = 0$ and $Z_2 = 0$ are observed, the probability of $\Pr(S_0[1] = 1 \wedge S_0[2] = 0 | Z_1 = 0 \wedge Z_2 = 0)$ is estimated as follows.

$$\begin{aligned} & \Pr(S_0[1] = 1 \wedge S_0[2] = 0 | Z_1 = 0 \wedge Z_2 = 0) \\ = & \frac{\Pr(S_0[1] = 1 \wedge S_0[2] = 0)}{Pr(Z_1 = 0 \wedge Z_2 = 0)} \\ & \cdot \Pr(Z_1 = 0 \wedge Z_2 = 0 | S_0[1] = 1 \wedge S_0[2] = 0) \\ = & \frac{\frac{1}{256} \cdot \frac{1}{255}}{2^{-14.427}} \cdot 1 = 2^{-1.567}. \end{aligned}$$

Thus, the event of $Z_1 = 0$ and $Z_2 = 0$ allows us to guess two bytes of the state of $S_0[1]$ and $S_0[2]$ with the probability of $2^{-1.567}$, while the probability of the random guess is 2^{-16}. This state guess has a $2^{14.433}$ times advantage over the random guess.

Moreover, we also guess the value of the key by using the following Roos's state-key relations [134].

$$S_0[1] = 1 + K[0] + K[1], \quad S_0[2] = 3 + K[0] + K[1] + K[2].$$

Above two 1-byte equations hold with the probability of 0.371 and 0.368, respectively [130]. Thus, when $Z_1 = 0$ and $Z_2 = 0$ are observed, the success probability of guessing above equations is $(0.371) \cdot (0.368) \cdot (2^{-1.567}) = 2^{-4.440}$. This key recovery has a $2^{11.560}$ times advantage over the random guess.

9.6.2 Key (State) Recovery Attack Using $Z_3 = 131$

When $Z_3 = 131$ is observed, the probability of $\Pr(S_0[1] = 1 \wedge S_0[2] = 0|Z_3 = 131)$ is estimated as follows.

$$\begin{aligned} & \Pr(S_0[1] = 131 \wedge S_0[2] = 128|Z_3 = 131) \\ = & \frac{\Pr(S_0[1] = 131 \wedge S_0[2] = 128)}{Pr(Z_3 = 131)} \\ & \cdot \Pr(Z_3 = 131|S_0[1] = 131 \wedge S_0[2] = 128) \\ = & \frac{\frac{1}{256} \cdot \frac{1}{255}}{2^{-7.99947}} \cdot 1 = 2^{-8.000}. \end{aligned}$$

Thus, the event of $Z_3 = 131$ allows us to guess two bytes of the state of $S_0[1]$ and $S_0[2]$ with the probability of $2^{-8.000}$, which has a $2^{8.000}$ times advantage over the random guess.

Similar to the key recovery attack using the conditional biases, Roos's state-key relations regarding $S_0[1]$ and $S_0[2]$ can be used. When $Z_1 = 0$ and $Z_2 = 0$ are observed, the success probability of guessing above key equations is $(0.371) \cdot (0.368) \cdot (2^{-8.000}) = 2^{-10.873}$. This key recovery has a $2^{5.128}$ times advantage over the random guess.

9.6.3 Key (State) Recovery Attack Using Extended Keylength Dependent Biases

When $Z_r = -r$ is observed, the probability of $\Pr(S_{r-1}[r] = 0 \mid Z_r = -r)$ is estimated as follows.

$$\Pr(S_{r-1}[r] = 0 \mid Z_r = -r) = \frac{\Pr(Z_r = -r \wedge S_{r-1}[r] = 0)}{\Pr(Z_r = -r)},$$

where $\Pr(Z_r = -r \wedge S_{r-1}[r] = 0)$ is given in Theorem 12 and $\Pr(Z_r = -r)$ is given in Theorem 11. If j does not touch the index r during first $r - 1$ rounds of the PRGA, $S_0[r] = 0$ is expected. The probability of $\Pr(S_0[r] = 0 \mid Z_r = -r)$ is estimated as follows,

$$\Pr(S_0[r] = 0 \mid Z_r = -r) = \frac{1}{N} + \left(1 - \frac{1}{N}\right)^r \cdot p_0,$$

where $p_0 = \Pr(S_{r-1}[r] = 0 \mid Z_r = -r)$. Table 9.4 shows experimental and theoretical values of $\Pr(S_0[r] = 0 \mid Z_r = -r)$. All cases are more efficient than the random guess.

The key recovery attack can be constructed by using the following Roos's state-key relation [134],

$$S_0[r] = \frac{r \cdot (r + 1)}{2} + \sum_{x=0}^{r} K[x].$$

For $r = 16$, above equation holds with the probability of 0.206 [130]. When $Z_{16} = -16$ is observed, the success probability of guessing above equation is $(0.206) \cdot$

$(2^{-4.772}) = 2^{-7.051}$. Thus, one byte of the key information is obtained with probability of $2^{-7.051}$.

Furthermore it can be improved by using the other event. When $Z_r = -r$ is observed, the probability of $\Pr(j_r^K = -r \mid Z_r = -r)$ is estimated as follows

$$\Pr(j_r^K = -r \mid Z_r = -r) = \frac{\Pr(j_r^K = -r)}{\Pr(Z_r = -r)} \cdot \Pr(Z_r = -r \mid j_r^K = -r),$$

where $\Pr(j_r^K = -r) = 1/N$ and $\Pr(Z_r = -r)$ is given in Theorem 11. $\Pr(Z_r = -r \mid j_r^K = -r)$ is approximately

$$\Pr(Z_r = -r \mid j_r^K = -r) \approx \frac{1}{N^2} + \left(1 - \frac{1}{N^2}\right)\gamma'_r + (1-\delta_r) \cdot \frac{1}{N},$$

where $\delta_r = \Pr(S_r[j_r] = 0) = \Pr(S_{r-1}[r] = 0)$ and

$$\gamma'_r = \frac{1}{N} \cdot \left(1 - \frac{r+1}{N}\right) \cdot \sum_{x=r+1}^{N-1} \left(1 - \frac{1}{N}\right)^x \cdot$$
$$\left(1 - \frac{2}{N}\right)^{x-r} \cdot \left(1 - \frac{3}{N}\right)^{N-x+2r-4}.$$

Here, δ_r is obtained from Theorem 6. Table 9.5 shows experimental and theoretical values of $\Pr(j_r^K = -r \mid Z_r = -r)$. The event of $Z_r = -r$ allows us to guess the value of j_r^K with the probability of $2^{-4.737}$ for $(r = 16)$. Using following Roos's key-correlation in RC4 [134], the key is also guessed.

$$j_r^K = -r = \frac{r(r-1)}{2} + \sum_{x=0}^{r-1} K[x].$$

When $r = 16$, above equation is satisfied with the probability of 0.627 [130]. Then, when $Z_{16} = -16$ is observed, the success probability of guessing above equation is $(0.627) \cdot (2^{-4.752}) = 2^{-5.426}$. Using it with the method in the previous event, two bytes of key information are guessed with the probability of $2^{-12.477}$. Therefore, this key recovery attack has a $2^{3.523}$ times advantage against the random key guess. Since the experimental value of this two bytes of key information obtained by a computer experiment for 2^{33} randomly-chosen keys is $2^{-13.14}$, this attack also can be experimentally confirmed.

However, the other cases such as $r = 32, 48, \ldots, 112$ have no advantage over the random guess for the key recovery.

9.7 (Multiple-Key) Distinguishing Attack

Our new biases are also applicable to distinguishing attacks, which distinguish an RC4 keystream from a random stream. The number of required samples to the successful distinguishing attack is given as the followings theorem.

Theorem 14. **[33]** *Let X and Y be two distributions, and suppose that the event e happens in X with probability p and in Y with probability $p \cdot (1+q)$. Then for small p and q, $O(\frac{1}{p \cdot q^2})$ samples suffice to distinguish X from Y with a constant probability of success.*

Table 9.4: Experimental and theoretical values of $\Pr(S_0[r] = 0 \mid Z_r = -r)$

r	Prob.(Theoretical)	Prob.(Experimental)
16	$2^{-4.676}$	$2^{-4.772}$
32	$2^{-5.177}$	$2^{-5.257}$
48	$2^{-5.640}$	$2^{-5.697}$
64	$2^{-6.060}$	$2^{-6.102}$
80	$2^{-6.416}$	$2^{-6.588}$
96	$2^{-6.706}$	$2^{-6.951}$
112	$2^{-6.931}$	$2^{-7.256}$

Table 9.5: Experimental and theoretical values of $\Pr(j_r^K = -r \mid Z_r = -r)$

r	Prob.(Theoretical)	Prob.(Experimental)
16	$2^{-4.737}$	$2^{-4.752}$
32	$2^{-5.217}$	$2^{-5.203}$
48	$2^{-5.686}$	$2^{-5.668}$
64	$2^{-6.134}$	$2^{-6.067}$
80	$2^{-6.549}$	$2^{-6.560}$
96	$2^{-6.919}$	$2^{-6.929}$
112	$2^{-7.232}$	$2^{-7.233}$

Multiple-key distinguishing attack aims to distinguishes the set of keystreams, generated by multiple keys, from a random stream. The known best multiple-key distinguishing attack is proposed by Mantin and Shamir [33]. Since this attack exploits the bias such that that second byte of a keystream is biased to 0, only second bytes of keystreams generated by different keys is used for the distinguishing attack. When $N = 256$, this attack requires about $\frac{1}{p \cdot q^2} = N = 2^8$ samples (the second byte of keystreams), where p and q are given as $p = 1/N$ and $q = 1$.

In order to get rid of such strong biases of initial bytes, the RC4-drop(n) is proposed as a countermeasure. It disregards first n bytes of the keystream of RC4 [2]. In the case of $n \geq 2$, the above multiple-key distinguishing attack using the second byte of keystreams does not work. Hereafter, we only deal with the typical parameter $N = 256$ for simplicity.

The keylength-dependent biases [133], such that ℓ-th bytes of keystreams is biased to $-\ell$, becomes the most efficient multiple-key distinguishing attack in the case of $2 \leq n \leq \ell - 1$ with respect to the number of samples. In the case of $\ell \leq n \leq 254$, the Maitra-Paul-Sen Gupta Bias [34] becomes the most efficient attack.

We improve these attacks by using our new biases in conjunction with known biases. For the multiple use of biases in each byte, the following lemma for the discrimination is given.

Lemma 5. **[37]** *Let X and Y be two distributions and suppose that the independent*

[2]RC4-drop(n) is a generalized implementation of the countermeasure written by [125], and this is defined at `http://www.users.zetnet.co.uk/hopwood/crypto/scan/cs.html`.

events $\{e_i\colon 1 \leq i \leq k\}$ occur with probabilities $\Pr_X(e_i) = p_i$ *in* X *and* $\Pr_Y(e_i) = (1 + q_i) \cdot p_i$ *in* Y. *Then the discrimination* D *of the distributions is* $\sum_i p_i \cdot q_i^2$.

The number of required samples for the distinguishing attack with the probability of $1 - \alpha$ is given as the following lemma.

Lemma 6. [37] *The number of samples that is required for distinguishing two distributions that have discrimination* D *with success rate* $1 - \alpha$ *(for both directions) is* $(1/D) \cdot (1 - 2\alpha) \cdot log_2 \frac{1-\alpha}{\alpha}$.

When the number of samples is $\frac{1}{p \cdot q^2}$, a success probability is estimated as about 0.778. For the fair comparison, we consider the distinguishing attack with success probability of $(1 - \alpha) = 0.778$.

Table 9.6 shows the number of required samples of multiple-key distinguishing attacks on RC4-drop(n), where each type of biases are Type A ($Z_2 = 0$ [33]), Type B ($Z_r = 0$ for $3 \leq r \leq 255$ [34, 35]), Type C ($Z_r = r$ for $3 \leq r \leq 255$ (our)), Type D ($Z_{16} = -240$ [36]), Type E ($Z_r = -r$ for $r = 32, 48, 64, 80, 96, 112$ (our)), Type F ($Z_{256} \neq 0$ (our)), Type G ($Z_{257} = 0$ (our)).. Note that our attack uses multiple biases, while previous attacks use only the single bias. These values are estimated by using Lemma 5 and 6 and the experimental values of biases obtained by 2^{40} tests. When $2 \leq n \leq 255$, our attack can improve the attack using previous single bias. In addition, when $n = 256$, our attacks is the first successful attack with the less samples than that of the known best single-key distinguishing attack, which require 2^{28} samples [37] [3].

Therefore, our multiple-key distinguishing attacks substantially improve the known best attack of RC4-drop(n).

9.8 Conclusion

In this chapter, we have evaluated the practical security of RC4 in the broadcast setting. After the introduction of four new biases of the keystream of RC4, i.e., the conditional bias of Z_1, the biases of $Z_3 = 131$ and $Z_r = r$ for $3 \leq r \leq 255$, and the extended keylength-dependent biases, a cumulative list of strongest known biases in $Z_1, Z_2, \ldots, Z_{257}$ is given. Then, we demonstrate a practical plaintext recovery attack using our bias set by a computer experiment. As a result, most bytes of $P_1, P_2, \ldots, P_{257}$ could be extracted with probability more than 0.8 using 2^{32} ciphertexts encrypted by randomly-chosen keys. Then, we have proposed an efficient method to extract bytes of plaintexts after P_{258}. Our attack is able to recover any plaintext byte from only ciphertexts generated using different keys. For example, first 2^{50} bytes of the plaintext are expected to be recovered from 2^{34} ciphertexts with high probability. Finally, we showed that our set of these biases are applicable to plaintext recovery attacks, key recovery attacks and distinguishing attacks.

Note that our attack on broadcast RC4, as proposed in this chapter, utilizes the advantage of sequential recovery of plaintext bytes. If the initial 256/512/768 bytes of the keystream are suppressed in the protocol, as recommended in case of RC4

[3]Paul and Preneel found the bias of $Z_{257} \neq Z_{258}$ and proposed a multiple-key distinguishing attack in FSE 2004 [126]. This attack works in $n = 256$, however the number of samples for success is around 2^{32}, which is larger than that of the known best single-key distinguishing attack [37].

Table 9.6: The number of required samples for multiple-key distinguishing attacks on RC4-drop(n)

Position of biases (r)	Range of best attacks	Multiple biases (our)		Single bias (previous)	
		#samples	Types	#samples	Types
2	$0 \leq n \leq 1$	2^{8}	A	2^{8}	A
16	$2 \leq n \leq 15$	$2^{17.58}$	B, C, D	$2^{17.63}$	D
32	$16 \leq n \leq 31$	$2^{18.69}$	B, C, E	$2^{23.76}$	B
48	$32 \leq n \leq 47$	$2^{19.76}$	B, C, E	$2^{24.13}$	B
64	$48 \leq n \leq 63$	$2^{20.82}$	B, C, E	$2^{24.50}$	B
80	$64 \leq n \leq 79$	$2^{22.10}$	B, C, E	$2^{24.82}$	B
96	$80 \leq n \leq 95$	$2^{23.22}$	B, C, E	$2^{25.14}$	B
112	$80 \leq n \leq 111$	$2^{24.25}$	B, C, E	$2^{25.48}$	B
113, 114 $\ldots, 183$	$n = r - 1$	min: $2^{25.17}$ max: $2^{26.84}$	B, C	min: $2^{25.51}$ max: $2^{26.92}$	B
184, 185 $\ldots, 255$	—	min: $2^{26.87}$ max: $2^{28.04}$	B, C	min: $2^{26.94}$ max: $2^{28.04}$	B
256	$183 \leq n \leq 255$	$2^{26.87}$	F	—	—
257	$n = 256$	$2^{27.02}$	G	—	—

usages [125], our attack does not work any more. However, widely-used protocols such as SSL/TLS use initial bytes of the keystream. For SSL/TLS, the broadcast setting is converted into the multi-session setting where the target plaintext block are repeatedly sent in the same position in the plaintexts in multiple SSL/TLS sessions [135].

Our evaluation reveals that broadcast RC4 is practically vulnerable to the plaintext recovery attacks as moderate amount of ciphertexts, i.e., 2^{24} to 2^{34} ciphertexts generated by different keys, leaks considerable information about the plaintext. Thus, RC4 is not to be recommended for the encryption in case of the typical broadcast setting and multi-session setting of SSL/TLS.

Part III

Analysis of Hash Function

Chapter 10

Preimage Attack on Tiger and SHA-2

This chapter shows new preimage attacks on reduced Tiger and SHA-2. Indesteege and Preneel presented a preimage attack on Tiger reduced to 13 rounds (out of 24) with a complexity of $2^{128.5}$. Our new preimage attack finds a one-block preimage of Tiger reduced to 16 rounds with a complexity of 2^{161}. The proposed attack is based on meet-in-the-middle attacks. It seems difficult to find "independent words" of Tiger at first glance, since its key schedule function is much more complicated than that of MD4 or MD5. However, we developed techniques to find independent words efficiently by controlling its internal variables. Surprisingly, the similar techniques can be applied to SHA-2 including both SHA-256 and SHA-512. We present a one-block preimage attack on SHA-256 and SHA-512 reduced to 24 (out of 64 and 80) steps with a complexity of 2^{240} and 2^{480}, respectively. To the best of our knowledge, our attack is the best known preimage attack on reduced-round Tiger and our preimage attack on reduced-step SHA-512 is the first result. Furthermore, our preimage attacks can also be extended to second preimage attacks directly, because our attacks can obtain random preimages from an arbitrary IV and an arbitrary target.

10.1 Introduction

Cryptographic hash functions play an important role in the modern cryptology. Many cryptographic protocols require a secure hash function which holds several security properties such as classical ones: collision resistance, preimage resistance and second preimage resistance. However, a lot of hash functions have been broken by collision attacks including the attacks on MD5 [13] and SHA-1 [12]. These hash functions are considered to be broken in theory, but in practice many applications still use these hash functions because they do not require collision resistance. However, (second) preimage attacks are critical for many applications including integrity checks and encrypted password systems. Thus analyzing the security of the hash function with respect to (second) preimage resistance is important, even if the hash function is already broken by a collision attack. However, the preimage resistance of hash functions has not been studied well.

Table 10.1: Summary of our results

Target	Attack (first or second preimage)	Attacked steps (rounds)	Complexity
Tiger (full 24 rounds)	first [40]	13	$2^{128.5}$
	first **(this chapter)**	16	2^{161}
	second [40]	13	$2^{127.5}$
	second **(this chapter)**	16	2^{160}
SHA-256 (full 64 steps)	first **(this chapter)**	24	2^{240}
	second **(this chapter)**	24	2^{240}
SHA-512 (full 80 steps)	first **(this chapter)**	24	2^{480}
	second **(this chapter)**	24	2^{480}

Tiger is a dedicated hash function producing a 192-bit hash value designed by Anderson and Biham in 1996 [38]. As a cryptanalysis of Tiger, at FSE 2006, Kelsey and Lucks proposed a collision attack on 17-round Tiger with a complexity of 2^{49} [136], where full-version Tiger has 24 rounds. They also proposed a pseudo-near collision attack on 20-round Tiger with a complexity of 2^{48}. This attack was improved by Mendel et al. at INDOCRYPT 2006 [137]. They proposed a collision attack on 19-round Tiger with a complexity of 2^{62}, and a pseudo-near collision attack on 22-round Tiger with a complexity of 2^{44}. Later, they proposed a pseudo-near-collision attack of full-round (24-round) Tiger with a complexity of 2^{44}, and a pseudo-collision (free-start-collision) attack on 23-round Tiger [138]. The above results are collision attacks and there is few evaluations of preimage resistance of Tiger. Indesteege and Preneel presented preimage attacks on reduced-round Tiger [40]. Their attack found a preimage of Tiger reduced to 13 rounds with a complexity of $2^{128.5}$.

In this chapter, we introduce a preimage attack on reduced-round Tiger. The proposed attack is based on meet-in-the-middle attacks [16]. In this attack, we needs to find independent words ("neutral words") in the first place. However, the techniques used for finding independent words of MD4 or MD5 cannot be applied to Tiger directly, since its key schedule function is much more complicated than that of MD4 or MD5. To overcome this problem, we developed new techniques to find independent words of Tiger efficiently by adjusting the internal variables. As a result, the proposed attack finds a preimage of Tiger reduced to 16 (out of 24) rounds with a complexity of about 2^{161}. Surprisingly, our new approach can be applied to SHA-2 including both SHA-256 and SHA-512. We present a preimage attack on SHA-256 and SHA-512 reduced to 24 (out of 64 and 80) steps with a complexity of about 2^{240} and 2^{480}, respectively. As far as we know, our attack is the best known preimage attack on reduced-round Tiger and our preimage attack on reduced-step SHA-512 is the first result. Furthermore, we show that our preimage attacks can also be extended to second preimage attacks directly and all of our attacks can obtain one-block preimages, because our preimage attacks can obtain random preimages from an arbitrary IV and an arbitrary target. These results are summarized in Table 10.1.

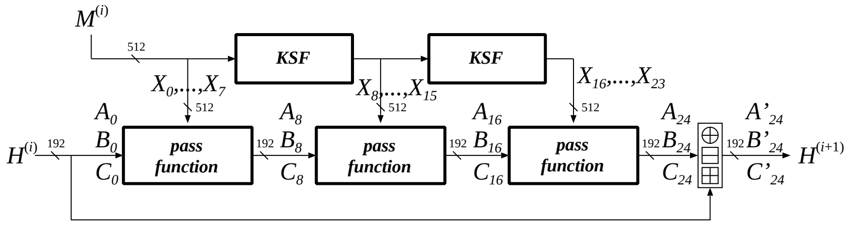


Figure 10.1: Compression function f of Tiger

10.2 Preliminaries

10.2.1 Description of Tiger

Tiger is an iterated hash function that compresses an arbitrary length message into a 192-bit hash value. An input message value is divided into 512-bit message blocks $(M^{(0)}, M^{(1)}, ..., M^{(t-1)})$ by the padding process as well as the MD family. The compression function of Tiger shown in Fig. 10.1 generates a 192-bit output chaining value $H^{(i+1)}$ from a 512-bit message block $M^{(i)}$ and a 192-bit input chaining value $H^{(i)}$ where chaining values consist of three 64-bit variables, $A_j^{(i)}$, $B_j^{(i)}$ and $C_j^{(i)}$. The initial chaining value $H^{(0)} = (A_0^{(0)}, B_0^{(0)}, C_0^{(0)})$ is as follows:

$$
\begin{aligned}
A_0^{(0)} &= \texttt{0x0123456789ABCDEF},\\
B_0^{(0)} &= \texttt{0xFEDCBA9876543210},\\
C_0^{(0)} &= \texttt{0xF096A5B4C3B2E187}.
\end{aligned}
$$

In the compression function, a 512-bit message block $M^{(i)}$ is divided into eight 64-bit words $(X_0, X_1, ..., X_7)$. The compression function consists of three pass functions and between each of them there is a key schedule function. Since each pass function has eight round functions, the compression function consists of 24 round functions. The pass function is used for updating chaining values, and the key schedule function is used for updating message values. After the third pass function, the following feedforward process is executed to give outputs of the compression function with input chaining values and outputs of the third pass function,

$$A'_{24} = A_0 \oplus A_{24},\ B'_{24} = B_0 - B_{24},\ C'_{24} = C_0 + C_{24},$$

where A_i, B_i and C_i denote the i-th round chaining values, respectively, and A'_{24}, B'_{24} and C'_{24} are outputs of the compression function.

In each round of the pass function, chaining values A_i, B_i and C_i are updated by a message word X_i as follows:

$$
\begin{aligned}
B_{i+1} &= C_i \oplus X_i, && (10.1)\\
C_{i+1} &= A_i - even(B_{i+1}), && (10.2)\\
A_{i+1} &= (B_i + odd(B_{i+1})) \times mul, && (10.3)
\end{aligned}
$$

where mul is the constant value $\in \{5, 7, 9\}$ which is different in each pass function.

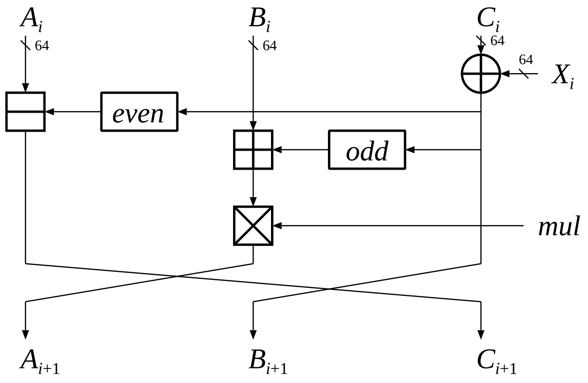


Figure 10.2: Tiger round function

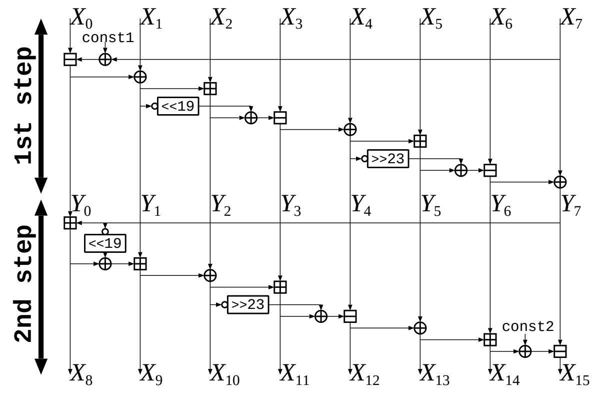

1st step
2nd step
X_0 X_1 X_2 X_3 X_4 X_5 X_6 X_7
const1
<<19
>>23
Y_0 Y_1 Y_2 Y_3 Y_4 Y_5 Y_6 Y_7
<<19
>>23
const2
X_8 X_9 X_{10} X_{11} X_{12} X_{13} X_{14} X_{15}

Figure 10.3: Key schedule function

The nonlinear functions *even* and *odd* are expressed as follows:

$$even(W) = T_1[w_0] \oplus T_2[w_2] \oplus T_3[w_4] \oplus T_4[w_6], \quad (10.4)$$

$$odd(W) = T_4[w_1] \oplus T_3[w_3] \oplus T_2[w_5] \oplus T_1[w_7], \quad (10.5)$$

where 64-bit value W is split into eight bytes $\{w_7, w_6, ..., w_0\}$ with w_7 is the most significant byte and $T_1, ..., T_4$ are the S-boxes: $\{0,1\}^8 \to \{0,1\}^{64}$. Figure 10.2 shows the round function of Tiger.

The key schedule function (KSF) updates message values. In the first pass function, eight message words $X_0, ..., X_7$, which are identical to input message blocks of the compression function, are used for updating chaining values. Remaining two pass functions use sixteen message words which are generated by applying KSF:

$$(X_8, ..., X_{15}) = KSF(X_0, ..., X_7), \quad (10.6)$$

$$(X_{16}, ..., X_{23}) = KSF(X_8, ..., X_{15}). \quad (10.7)$$

The function KSF which updates the inputs $X_0, ..., X_7$ in two steps. The first step shown in the left table generates internal variables $Y_0, ..., Y_7$ from inputs $X_0, ..., X_7$

as follows.

$$\begin{aligned}
Y_0 &= X_0 - (X_7 \oplus \texttt{const1}), && (10.8)\\
Y_1 &= X_1 \oplus Y_0, && (10.9)\\
Y_2 &= X_2 + Y_1, && (10.10)\\
Y_3 &= X_3 - (Y_2 \oplus (\overline{Y_1} \ll 19)), && (10.11)\\
Y_4 &= X_4 \oplus Y_3, && (10.12)\\
Y_5 &= X_5 + Y_4, && (10.13)\\
Y_6 &= X_6 - (Y_5 \oplus (\overline{Y_4} \gg 23)), && (10.14)\\
Y_7 &= X_7 \oplus Y_6. && (10.15)\\
X_8 &= Y_0 + Y_7, && (10.16)
\end{aligned}$$

The second step shown in the right table calculates outputs $X_8, ..., X_{15}$ from internal variables $Y_0, .., Y_7$ as follows.

$$\begin{aligned}
X_9 &= Y_1 - (X_8 \oplus (\overline{Y_7} \ll 19)), && (10.17)\\
X_{10} &= Y_2 \oplus X_9, && (10.18)\\
X_{11} &= Y_3 + X_{10}, && (10.19)\\
X_{12} &= Y_4 - (X_{11} \oplus (\overline{X_{10}} \gg 23)), && (10.20)\\
X_{13} &= Y_5 \oplus X_{12}, && (10.21)\\
X_{14} &= Y_6 + X_{13}, && (10.22)\\
X_{15} &= Y_7 - (X_{14} \oplus \texttt{const2}), && (10.23)
\end{aligned}$$

where `const1` is `0xA5A5A5A5A5A5A5A5` and `const2` is `0x0123456789ABCDEF`. By using the same function, $X_{16}, ..., X_{23}$ are also derived from $X_8, ..., X_{15}$. Figure. 10.3 shows the key schedule function of Tiger .

10.2.2 Description of SHA-256

We only show the structure of SHA-256, since SHA-512 is structurally very similar to SHA-256 except for the number of steps, word size and rotation values. The compression function of SHA-256 consists of a message expansion function and a state update function. The message expansion function expands 512-bit message block into 64 32-bit message words $W_0, ..., W_{63}$ as follows:

$$W_i = \begin{cases} M_i & (0 \le i < 16), \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & (16 \le i < 64), \end{cases}$$

where the functions $\sigma_0(X)$ and $\sigma_1(X)$ are given by

$$\begin{aligned}
\sigma_0(X) &= (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3), \\
\sigma_1(X) &= (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10).
\end{aligned}$$

The state update function updates eight 32-bit chaining values, $A, B, ..., G, H$ in 64 steps as follows:

$$\begin{aligned}
T_1 &= H_i + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + K_i + W_i, & (10.24)\\
T_2 &= \Sigma_0(A_i) + Maj(A_i, B_i, C_i), & (10.25)\\
A_{i+1} &= T_1 + T_2, & (10.26)\\
B_{i+1} &= A_i, & (10.27)\\
C_{i+1} &= B_i, & (10.28)\\
D_{i+1} &= C_i, & (10.29)\\
E_{i+1} &= D_i + T_1, & (10.30)\\
F_{i+1} &= E_i, & (10.31)\\
G_{i+1} &= F_i, & (10.32)\\
H_{i+1} &= G_i, & (10.33)
\end{aligned}$$

where K_i is a step constant and the function Ch, Maj, Σ_0 and Σ_1 are given as follows:

$$\begin{aligned}
Ch(X, Y, Z) &= XY \oplus \overline{X}Z,\\
Maj(X, Y, Z) &= XY \oplus YZ \oplus XZ,\\
\Sigma_0(X) &= (X \ggg 2) \oplus (X \ggg 13) \oplus (X \ggg 22),\\
\Sigma_1(X) &= (X \ggg 6) \oplus (X \ggg 11) \oplus (X \ggg 25).
\end{aligned}$$

After 64 step, a feedfoward process is executed with initial state variable by using word-wise addition modulo 2^{32}.

10.2.3 Meet-in-the-Middle Approach for Preimage Attack

We assume that a compression function F consists of a key scheduling function (KSF) and a round/step function as shown in Fig. 10.4. The function F has two inputs, an n-bit chaining variable H and an m-bit message M, and outputs an n-bit chaining variable G. The function KSF expands the message M, and provides them into the round/step function.

We consider a problem that given H and G, find a message M satisfying $G = F(H, M)$. This problem corresponds to the preimage attack on the compression function with a fixed input chaining variable. In this model, a feedforward function does not affect the attack complexity, since the targets H and G are arbitrary values. If we obtain a preimage from arbitrary values of H and G, we can also compute a preimage from H and $H \oplus G$ instead of G.

In the meet-in-the-middle preimage attack, we first divide the round function into two parts: the forward process (FP) and the backward process (BP) so that each process can compute an ℓ-bit meet point S independently. We also need independent words X and Y in KSF to compute S independently. The meet point S can be determined from FP and BP independently such that $S = FP(H, X)$ and $S = BP(G, Y)$.

If there are such two processes FP and BP, and independent words X and Y, we can obtain a message M satisfying S with a complexity of $2^{\ell/2}$ F evaluations,

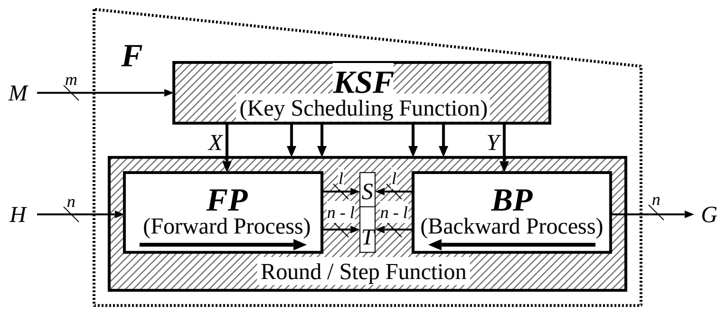


Figure 10.4: Meet-in-the-middle approach

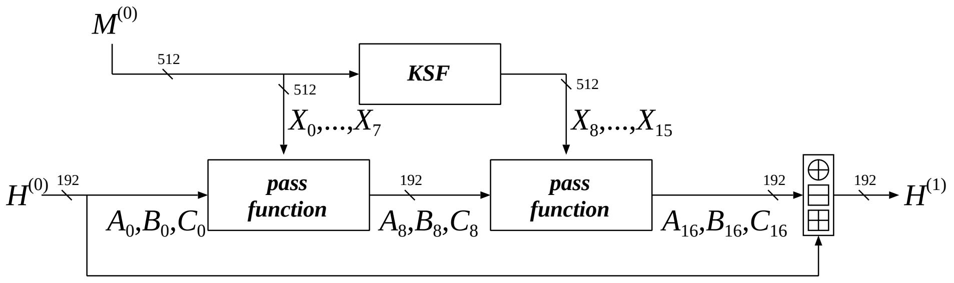


Figure 10.5: Reduced-round Tiger (2-pass = 16-round)

assuming that FP and BP are random ones, and the computation cost of BP is almost same as that of inverting function of BP. Since remaining internal state value T is $(n-\ell)$ bits, the desired M can be obtained with a complexity of $2^{n-\ell/2}(=2^{n-\ell+\ell/2})$. Therefore, if FP and BP up to the meet point S can be calculated independently, a preimage attack can succeed with a complexity of $2^{n-\ell/2}$. This type of preimage attacks on MD4 and MD5 was presented by Aoki and Sasaki [16].

In general, it is difficult to find such independent words in a complicated KSF. We developed new techniques to construct independent transforms in KSF by controlling internal variabes to obtain independent words.

10.3 Preimage Attack on Reduced-Round Tiger

In this section, we propose a preimage attack on 16-round Tiger with a complexity of 2^{161}. This variant shown in Fig. 10.5 consists of two pass functions and one key schedule function. First, we show properties of Tiger which are used for applying the meet-in-the-middle attack. Next, we show how to apply the meet-in-the-middle attack to Tiger, and then introduce the algorithm of our attack. Finally, we evaluate the required complexity and memory of our attack.

10.3.1 Properties of Tiger

We show five properties of Tiger, which enable us to apply the meet-in-the-middle attack.

Property 1 : *The pass function is easily invertible.*

Property 1 can be obtained from the design of the round function. From Eq. (10.1) to Eq. (10.3), A_i, B_i, and C_i can be determined from A_{i+1}, B_{i+1}, C_{i+1} and X_i. The computation cost is almost same as the cost of calculating A_{i+1}, B_{i+1} and C_{i+1} from A_i, B_i, C_i and X_i. Since the round function is invertible, we can construct the inverse pass function.

Property 2 : *In the inverse pass function, the particular message words are independent of particular state value.*

The detail of the Property 2 is that once X_i, A_{i+3} B_{i+3} and C_{i+3} are fixed, then C_i, B_{i+1}, A_{i+2} and B_{i+2} can be determined from Eq. (10.1) to Eq. (10.3) independently of X_{i+1} and X_{i+2}. Thus the property 2 implies that X_{i+1} and X_{i+2} are independent of C_i in the inverse pass function.

Property 3 : *In the round function, C_{i+1} is independent of odd bytes of X_i .*

The property 3 can be obtained from the property of the non-linear function *even*.

Property 4 :*The key schedule function KSF is easily invertible.*

The property 4 implies that we can build the inverse key schedule function KSF^{-1}. Moreover, the computation cost of KSF^{-1} is almost the same as that of KSF.

Property 5 :*In the inverse key schedule function KSF^{-1}, if input values are chosen appropriately, there are two independent transforms.*

The property 5 is one of the most important properties for our attack. In the next section, we show this in detail.

10.3.2 How to Obtain Two Independent Transforms in the KSF^{-1}

Since any input word of KSF^{-1} affects all output words of KSF^{-1}, it appears that there is no independent transform in the KSF^{-1} at first glance.

However, we analyzed the relation among the inputs and the outputs of KSF^{-1} deeply, and then found a technique to construct two independent transforms in the KSF^{-1} by choosing inputs carefully and controlling internal variables. Specifically, we can show that a change of input word X_8 only affects output words X_0, X_1, X_2 and X_3, and also modifications of X_{13}, X_{14} and X_{15} only affect X_5 and X_6 if these input words are chosen properly. We present the relation among inputs, outputs and internal variables of KSF^{-1} and then show how to build independent transforms in the KSF^{-1}.

As shown in Fig. 10.6, changes of inputs X_{13}, X_{14} and X_{15} only propagate internal variables Y_0, Y_1, Y_5, Y_6 and Y_7. If internal variables Y_6 and Y_7 are fixed even when X_{13}, X_{14} and X_{15} are changed, it can be considered that an internal variable Y_0, Y_1 and an output X_7 are independent of changes of X_{13}, X_{14} and X_{15}. From Eq. (10.22) and (10.23), Y_6 and Y_7 can be fixed to arbitrary values by choosing X_{13}, X_{14} and X_{15} satisfying the following formulae:

$$X_{14} = Y_6 + X_{13}, \qquad (10.34)$$
$$X_{15} = Y_7 - (X_{14} \oplus \texttt{const2}). \qquad (10.35)$$

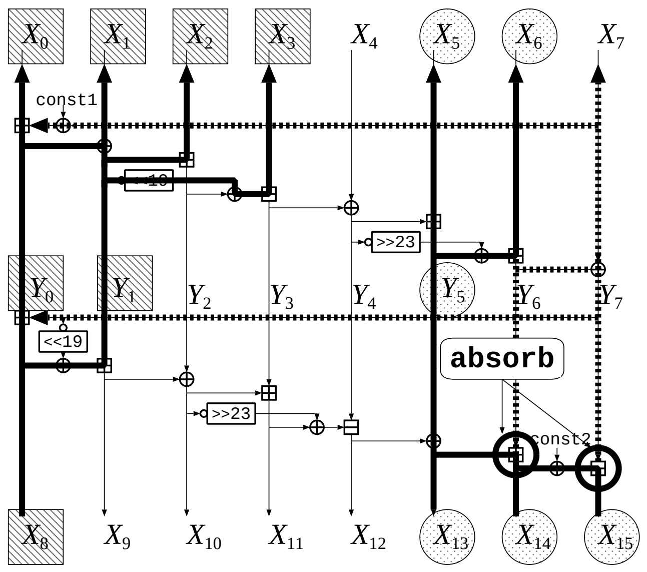


Figure 10.6: Relation among inputs and outputs of KSF^{-1}

Therefore modifications of inputs X_{13}, X_{14} and X_{15} only propagate X_5 and X_6 by selecting these input values appropriately. In addition, a modification of X_8 only affects X_0, ..., X_3.

As a result, we obtain two independent transforms in KSF^{-1} by choosing X_{13}, X_{14} and X_{15} properly, since in this case a change of X_8 only affects X_0, ..., X_3, and changes of X_{13}, X_{14} and X_{15} only propagate X_5 and X_6.

10.3.3 Applying Meet-in-the-Middle Attack to Reduced-Round Tiger

We show the method for applying the meet-in-the-middle attack to Tiger by using above five properties. We define the meet point as 64-bit C_6, the process 1 as rounds 1 to 6, and the process 2 as rounds 7 to 16.

In the process 2, intermediate values A_9, B_9 and C_9 can be calculated from A_{16}, B_{16}, C_{16} and message words X_9 to X_{15}, since Tiger without the feedforward function is easily invertible. From the property 2, C_6 can be determined from A_8, B_8 and X_6. It is also observed that A_8 and B_8 are independent of X_8, because these values are calculated from A_9, B_9 and C_9. From the property 5, X_8 does not affect X_6. Therefore, C_6, the output of the process 2, can be determined from X_6, X_9 to X_{15}, A_{16}, B_{16} and C_{16}.

In the process 1, the output C_6 can be calculated from X_0 to X_5, A_0, B_0 and C_0. If some changes of the message words used in each process do not affect the message words used in the other process, C_6 can be determined independently in each process.

The message words X_0 to X_4 are independent of changes of X_6 and X_{13} to X_{15}, if X_9 to X_{12} are fixed and X_{13} to X_{15} are calculated as illustrated in the section 10.3.2. Although changes of X_{13}, X_{14} and X_{15} propagate X_5, from the property 3, C_6 in the process 1 is not affected by changes of odd bytes of X_5. Therefore, if even bytes of X_5 are fixed, C_6 in the process 1 can be determined independently from a change

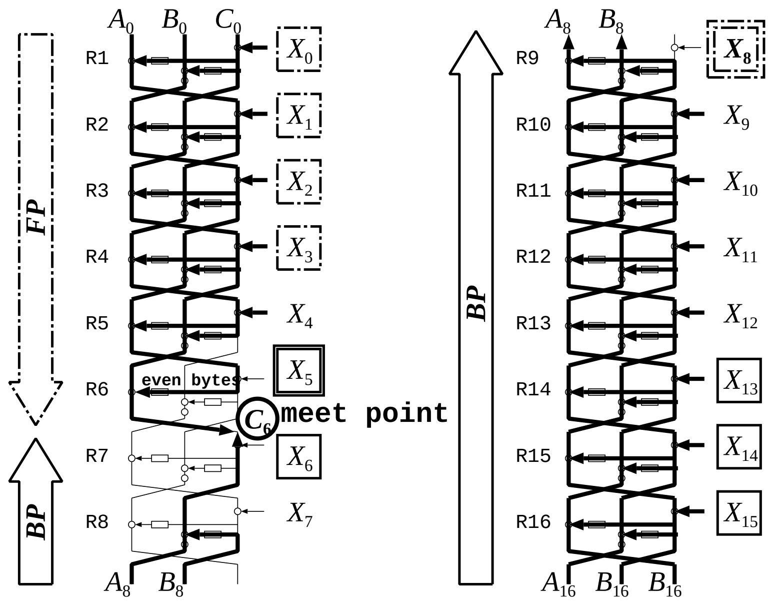


Figure 10.7: Meet-in-the-middle attack on 16-round Tiger

of X_5.

We show that the even bytes of X_5 can be fixed by choosing X_{11}, X_{12} and X_{13} properly. From Eq. (10.21), Y_5 is identical to X_{13} when X_{12} equals zero, and from Eq. (10.13), X_5 is identical to Y_5 when Y_4 equals zero. Thus X_5 is identical to X_{13} when both X_{12} and Y_4 are zero. Consequently, if the even bytes of X_{13} are fixed, and X_{12} and Y_4 equal zero, the even bytes of X_5 can be fixed. Y_4 can be fixed to zero by choosing X_{11} as $X_{11} \leftarrow \overline{X_{10}} \gg 23$. Therefore, if the following conditions are satisfied, C_6 in the process 1 can be independent of changes of X_{13}, X_{14} and X_{15}.

- X_9 and X_{10} are fixed arbitrarily,
- $X_{11} = \overline{X_{10}} \gg 23,\ X_{12} = 0$,
- X_{13}, X_{14} and X_{15} are chosen properly.

By choosing inputs of the inverse pass function satisfying the above conditions, we can execute the process 1 and the process 2 independently. Specifically, if only X_{13}, X_{14} and X_{15} are treated as variables in the process 2, then the process 2 can be executed independently from the process 1. Similarly, if only X_8 is treated as a variable in the process 1, then the process 1 is independent of the process 2, as long as X_8 to X_{15} satisfy the above conditions. These results are shown in Fig. 10.7.

10.3.4 (Second) Preimage Attack on 16-Round Tiger Compression Function

We present the whole algorithm of the (second) preimage attack on the compression function of Tiger reduced to 16 rounds. The attack consists of three phases: preparation, first and second phase.

The preparation phase sets $X_i (i \in \{4, 7, 9, 10, 11, 12\})$, $Y_i (i \in \{2, 3, 4, 6, 7\})$ and even bytes of X_{13} as follows:

Preparation

1: Let A'_{16}, B'_{16} and C'_{16} be given targets. Choose A_0, B_0 and C_0 arbitrarily, and set A_{16}, B_{16} and C_{16} as follows:

$$A_{16} \leftarrow A_0 \oplus A'_{16},\ B_{16} \leftarrow B_0 - B'_{16},\ C_{16} \leftarrow C'_{16} - C_0.$$

2: Choose X_9, X_{10}, Y_6, Y_7 and even bytes of X_{13} arbitrarily, set X_{12} and Y_4 to zero, and set X_7, X_{11}, Y_2, Y_3 and X_4 as follows:

$$X_7 \leftarrow Y_6 \oplus Y_7,\ X_{11} \leftarrow \overline{X_{10}} \gg 23,\ Y_2 \leftarrow X_9 \oplus X_{10},\ Y_3 \leftarrow X_{11} - X_{10},\ X_4 \leftarrow Y_3.$$

The first phase makes a table of (C_6, odd bytes of X_{13}) pairs in the process 2 as follows:

First Phase

1: Choose odd bytes of X_{13} randomly.

2: Set X_5, X_6, X_{14} and X_{15} as follows:

$$X_5 \leftarrow X_{13},\ X_6 \leftarrow Y_6 + X_{13},\ X_{14} \leftarrow Y_6 + X_{13},\ X_{15} \leftarrow Y_7 - ((Y_6 + X_{13}) \oplus \texttt{const2}).$$

3: Compute C_6 from $A_{16}, B_{16}, C_{16}, X_6$ and X_9 to X_{15}.

4: Place a pair (C_6, odd bytes of X_{13}) into a table.

5: If all 2^{32} possibilities of odd bytes of X_{13} have been checked, terminate this phase. Otherwise, set another value, which has not been set yet, to odd bytes of X_{13} and return to the step 2.

The second phase finds the desired message values X_0 to X_{15} in the process 1 by using the table as follows:

Second Phase

1: Choose X_8 randomly.

2: Set Y_0, Y_1, X_0, X_1, X_2 and X_3 as follows:

$$\begin{aligned}
Y_0 &\leftarrow X_8 - X_7,\\
Y_1 &\leftarrow X_9 + (X_8 \oplus (\overline{Y_7} \ll 19)),\\
X_0 &\leftarrow Y_0 + (X_7 \oplus \texttt{const1}),\\
X_1 &\leftarrow Y_0 \oplus Y_1,\\
X_2 &\leftarrow Y_2 - Y_1,\\
X_3 &\leftarrow Y_3 + (Y_2 \oplus (\overline{Y_1} \ll 19)).
\end{aligned}$$

3: Compute C_6 from X_0 to X_4, even bytes of X_5, A_0, B_0 and C_0.

4: Check whether this C_6 is in the table generated in the first phase. If C_6 is in the table, the corresponding X_0 to X_7 are a preimage for the compression function of the target $A'_{16}, B'_{16}, C'_{16}$ and successfully terminates the attack. Otherwise, set another value, which has not been set yet, to X_8 and return to the step 2.

By repeating the second phase about 2^{32} times for different choices of X_8, we expect to obtain a matched C_6. The complexity of the above algorithm is $2^{32}(= 2^{32} \cdot \frac{6}{16} + 2^{32} \cdot \frac{10}{16})$ compression function evaluations, and success probability is about 2^{-128}. By executing the above algorithm 2^{128} times with different fixed values, we can obtain a preimage of the compression function. In the preparation phase, A_0, B_0, C_0, X_9, X_{10}, Y_6, Y_7 and even bytes of X_{13} can be chosen arbitrarily. In other words, this attack can use these values as free words. These free words are enough for searching 2^{128} space. Accordingly, the complexity of the preimage attack on the compression function is $2^{160}(= 2^{32} \cdot 2^{128})$. Also, this algorithm requires 2^{32} 96-bit or $2^{35.6}$ bytes memory.

10.3.5 One-Block (Second) Preimage Attack on 16-Round Tiger

The preimage attack on the compression function can be extended to the one-block preimage attack on 16-round Tiger hash function. For extending the attack, A_0, B_0, C_0 are fixed to the IV words, the padding word X_7 is fixed to 447 encoded in 64-bit string, and the remaining 224 bits are used as free bits in the preparation phase. Although our attack cannot deal with another padding word X_6, the attack still works when the least significant bit of X_6 equals one.

Hence, the success probability of the attack on the hash function is half of that of the attack on the compression function. The total complexity of the one-block preimage attack on 16-round Tiger hash function is 2^{161} compression function computations.

This preimage attack can also be extended to the one-block second preimage attack directly. Our second preimage attack obtains a one-block preimage with the complexity of 2^{161}. Moreover, the complexity of our second preimage attack can be reduced by using the technique given in [40]. In this case, the second preimage attack obtains the preimage which consists of at least two message blocks with a complexity of 2^{160}.

10.4 Preimage Attack on Reduced-Round SHA-2

We apply our techniques to SHA-2 including both SHA-256 and SHA-512 in straightforward and present a preimage attack on SHA-2 reduced to 24 (out of 64 and 80, respectively) steps. We first check the properties of SHA-2, then introduce the algorithm of the preimage attack on 24-step SHA-2.

10.4.1 Properties of 24-step SHA-2

We first check whether SHA-2 has similar properties of Tiger. The pass function of Tiger corresponds to the 16-step state update function of SHA-2, and the key

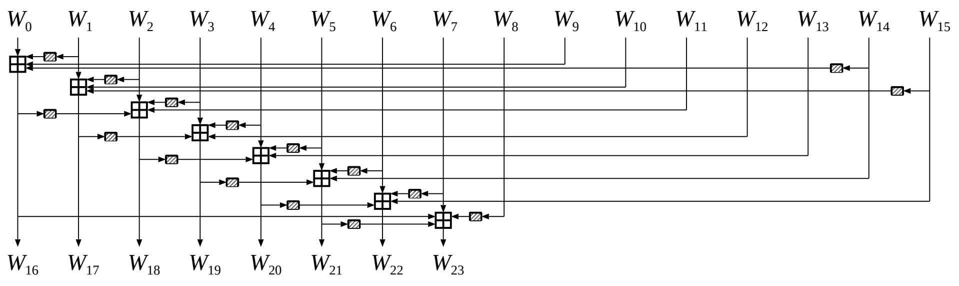

W_0 W_1 W_2 W_3 W_4 W_5 W_6 W_7 W_8 W_9 W_{10} W_{11} W_{12} W_{13} W_{14} W_{15}
W_{16} W_{17} W_{18} W_{19} W_{20} W_{21} W_{22} W_{23}

Figure 10.8: Message expansion function of 24-step SHA-2

Table 10.2: Relation among message values W_{16} to W_{23}.

computed value	values for computing
W_{16}	W_{14}, W_9, W_1*, W_0*
$W_{17}*$	$W_{15}, W_{10}, W_2, W_1*$
W_{18}	$W_{16}, W_{11}*, W_3*, W_2$
$\underline{W_{19}}$	$W_{17}*, \underline{W_{12}}, W_4, W_3*$
W_{20}	W_{18}, W_{13}, W_5, W_4
$\underline{W_{21}}$	$\underline{W_{19}}, W_{14}, W_6, W_5$
W_{22}	W_{20}, W_{15}, W_7, W_6
$\underline{W_{23}}$	$\underline{W_{21}}, W_{16}, W_8, W_7$

schedule function of Tiger corresponds to the 16-step message expansion function of SHA-2. Since the state update function and the message expansion function of SHA-2 are easily invertible, the compression function of SHA-2 without the feedforward function is also invertible.

In the inverse state update function, $A_{18}, B_{18}, ..., H_{18}$ are determined from $A_{24}, B_{24}, ..., H_{24}$ and W_{18} to W_{23}, and A_{11} only depends on $A_{18}, ..., H_{18}$. Thus A_{11} is independent of W_{11} to W_{17} when $A_{18}, ..., H_{18}$ and W_{18} to W_{23} are fixed. It corresponds to the property 2 of Tiger.

Then we check whether there are independent transforms in the inverse message expansion function of SHA-2. It corresponds to the property 5 of Tiger. For the 24-step SHA-2, 16 message words W_0 to W_{15} used in the first 16 steps are identical to input message blocks of the compression function, and 8 message words W_{16} to W_{23} used in the remaining eight steps are derived from W_0 to W_{15} by the message expansion function shown in Fig. 10.8. Table 10.2 shows the relation among message words in the message expansion function. For example, W_{16} is determined from W_{14}, W_9, W_1 and W_0. By using these relation and techniques introduced in previous sections, we can configure two independent transforms in the message expansion function of SHA-2.

We show that, in the inverse message expansion function of 24-step SHA-2, i) a change of W_{17} only affects W_0, W_1, W_3 and W_{11}, and ii) W_{19}, W_{21} and W_{23} only affect W_{12} by using the message modification techniques. In Tab. 10.2, asterisked values are variables of i), and underlined values are variables of ii).

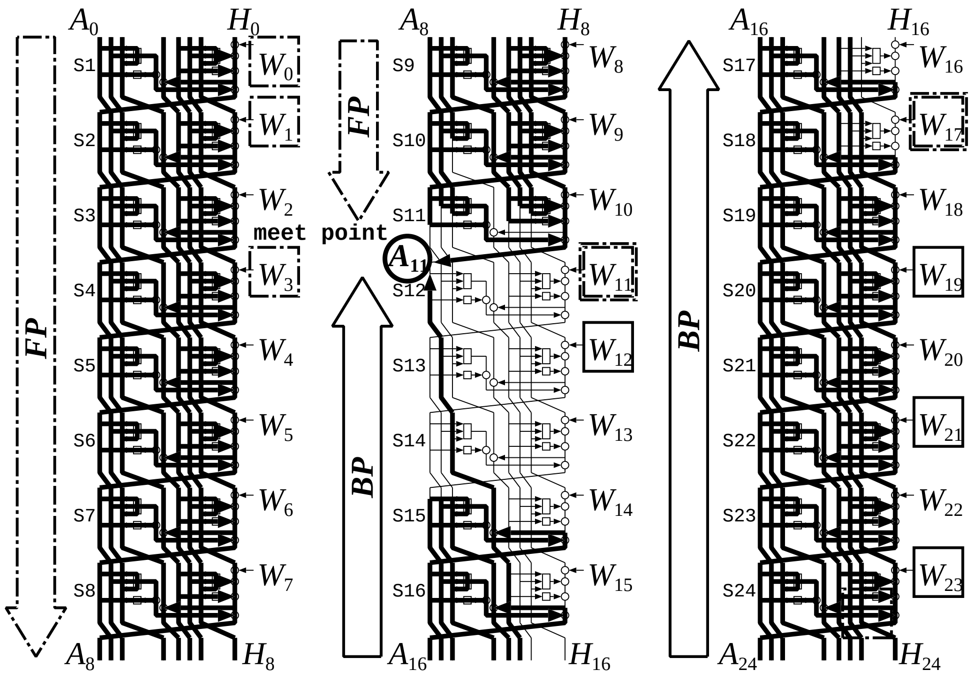


Figure 10.9: Meet-in-the-middle attack on 24-step SHA-2

First, we consider the influence of W_{23}. Though W_{23} affects W_7, W_8, W_{16} and W_{21}, this influence can be absorbed by modifying $W_{21} \to W_{19} \to W_{12}$. Consequently, we obtain a result that W_{19}, W_{21} and W_{23} only affect W_{12} by choosing these values properly, since W_{12} does not affect any other values in the inverse message expansion function.

Similarly, we consider the influence of W_{17} in the inverse message expansion function. W_{17} affects W_1, W_2, W_{10} and W_{15}. This influence can be absorbed by modifying $W_1 \to W_0$. W_{17} is also used for generating W_{19}. In order to cancel this influence, $W_3 \to W_{11}$ are also modified. As a result, we obtain a result that W_{17} only affects W_0, W_1, W_3 and W_{11} by choosing these values appropriately.

10.4.2 (Second) Preimage Attack on 24-Step SHA-256 Compression Function

As shown in Fig. 10.9, we define the meet point as 32-bit A_{11}, the process 1 as steps 1 to 11, and the process 2 as steps 12 to 24. In the process 1, A_{11} can be derived from $A_0, ..., H_0$ and W_0 to W_{10}. Similarly, in the process 2, A_{11} can be determined from $A_{24}, ..., H_{24}$ and W_{18} to W_{23}. Since the process 1 and process 2 are independent of each other for A_{11} by using the above properties of SHA-2, we apply the meet-in-the-middle attack to SHA-2 as follows:

Preparation

1: Let $A'_{24}, ..., H'_{24}$ be given targets. Choose $A_0, ..., H_0$ arbitrarily, and compute $A_{24}, ..., H_{24}$ by the feedforward function.

2: Choose 32-bit value `CON` and $W_i (i \in \{2, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16, 18\})$ arbitrarily, and then calculate W_{20} and W_{22}.

First Phase

1: Choose W_{23} randomly.

2: Determine W_{21}, W_{19} and W_{12} as follows[1]:

$$\begin{aligned} W_{21} &\leftarrow \sigma_1^{-1}(W_{23} - W_{16} - \sigma_0(W_8) - W_7), \\ W_{19} &\leftarrow \sigma_1^{-1}(W_{21} - W_{14} - \sigma_0(W_6) - W_5), \\ W_{12} &\leftarrow W_{19} - \mathtt{CON}. \end{aligned}$$

3: Compute A_{11} from $A_{24}, ..., H_{24}$ and W_{18} to W_{23}.

4: Place a pair (A_{11}, W_{23}) into a table.

5: If 2^{16} pairs of (A_{11}, W_{23}) have been listed in the table, terminate this algorithm. Otherwise, set another value, which has not been set yet, to W_{23} and return to the step 2.

Second Phase

1: Choose W_{17} randomly.

2: Determine W_0, W_1, W_3 and W_{11} as follows:

$$\begin{aligned} W_1 &\leftarrow W_{17} - \sigma_1(W_{15}) - W_{10} - \sigma_0(W_2), \\ W_0 &\leftarrow W_{16} - \sigma_1(W_{14}) - W_9 - \sigma_0(W_1), \\ W_3 &\leftarrow \mathtt{CON} - \sigma_1(W_{17}) - \sigma_0(W_4), \\ W_{11} &\leftarrow W_{18} - \sigma_1(W_{16}) - \sigma_0(W_3) - W_2. \end{aligned}$$

3: Compute A_{11} from $A_0, ..., H_0$ and W_0 to W_{10}.

4: Check whether this A_{11} is in the table generated in the first phase. If A_{11} is in the table, the corresponding W_0 to W_{23} is a preimage of the compression function of the target $A'_{24}, ..., H'_{24}$ and successfully terminates the attack. Otherwise, set another value, which has not been set yet, to W_{17} and return to the step 2.

By repeating the second phase about 2^{16} times for different W_{17}, we expect to obtain a matched A_{11}. The complexity of the preimage attack on the compression function is $2^{240} (= 2^{256-32/2})$ compression function evaluations. The required memory is 2^{16} 64-bit or 2^{19} bytes. In this attack, the words $A_0, ..., H_0$, `CON` and $W_i (i \in \{2, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15, 16, 18\})$ can be used as free words. The total free words are 22 words or 704 bits.

[1]The method how to calculate σ_1^{-1} is illustrated in the appendix.

10.4.3 One-Block (Second) Preimage Attack on 24-Step SHA-2 Hash Function

The preimage attack on the compression function can be extended to the (second) preimage attack on the hash function directly, since our preimage attack can obtain random preimages from an arbitrary IV and an arbitrary target, and can deal with the padding words W_{14} and W_{15}. Thus the complexities of the preimage attack and the second preimage attack on 24-step SHA-256 are 2^{240}. Furthermore, this attack can also be extended to the (second) preimage attack on 24-step SHA-512. The complexities of the (second) preimage attack on 24-step SHA-512 are $2^{480}(= 2^{512-64/2})$.

10.5 Conclusion

In this chapter, we have shown preimage attacks on reduced-round Tiger, reduced-step SHA-256 and reduced-step SHA-512. The proposed attacks are based on meet-in-the-middle attack. We developed new techniques to find "independent words" of the compression functions. In the attack on reduced-round Tiger, we found the "independent transforms" in the message schedule function by adjusting the internal variables, then we presented there are independent words in the compression function of Tiger. In the attack on reduced-round SHA-2, we found the "independent transforms" in the message expansion function by modifying the messages, then we showed that there are independent words in the compression function of SHA-2.

Our preimage attack can find a preimage of 16-step Tiger, 24-step SHA-256 and 24-step SHA-512 with a complexity of 2^{161}, 2^{240} and 2^{480}, respectively. These preimage attacks can be extended to second preimage attacks with the almost same complexities. Moreover, our (second) preimage attacks can find a one-block preimage, since it can obtain random preimages from an arbitrary IV an arbitrary target, and can also deal with the padding words.

Chapter 11

Converting Meet-In-The-Middle Preimage Attack into Pseudo Collision Attack: Application to SHA-2

In this chapter, we present a new technique to construct a collision attack from a particular preimage attack which is called a partial target preimage attack. Since most of the recent meet-in-the-middle preimage attacks can be regarded as the partial target preimage attack, a collision attack is derived from the meet-in-the-middle preimage attack. By using our technique, pseudo collisions of the 43-step reduced SHA-256 and the 46-step reduced SHA-512 can be obtained with complexities of 2^{126} and $2^{254.5}$, respectively. As far as we know, our results are the best pseudo collision attacks on both SHA-256 and SHA-512 in literature. Moreover, we show that our pseudo collision attacks can be extended to 52 and 57 steps of SHA-256 and SHA-512, respectively, by combined with the recent preimage attacks on SHA-2 by bicliques. Furthermore, since the proposed technique is quite simple, it can be directly applied to other hash functions. We apply our algorithm to several hash functions including Skein which is the SHA-3 finalists. We present not only the best pseudo collision attacks on SHA-2 family, but also a new insight of relation between a meet-in-the-middle preimage attack and a pseudo collision attack.

11.1 Introduction

Cryptographic hash functions play a central role in the modern cryptography. A secure hash function, which produces a fixed length hash value from an arbitrary length message, is required to satisfy at least three security properties: preimage resistance, second preimage resistance and collision resistance.

While there has not been a generic method to convert a collision attack into a preimage attack, it has been known that the preimage attack that can find at least two distinct preimages from the same target can be directly converted into a collision attack. However, the converted collision attack is often not efficient due to that the birthday bound of a collision attack ($2^{n/2}$) is far lower than the generic bound of the preimage attack (2^n), where n is the bit size of the hash value. Thus, it is left as open question that how to convert an efficient preimage attack into an efficient collision attack. In the case of the reduced SHA-256 regarding the number of attacked rounds, a preimage attack, covering 43 steps [84], is much better than the best known collision attack, with only 27 steps [139]. Moreover, basically, a collision attack and a preimage attack require quite different techniques. In other words, in general, the techniques used for the collision attack do not work well for a preimage attack, and vice versa. In fact, most of the recent collision attacks are based on a differential attack [13, 12], in contrast to that most of the recent preimage attacks are based on a meet-in-the-middle (MITM) attack [16]. Though converting the differential collision attack to a (pseudo) preimage attack was discussed in [140], there is no generic way to construct a collision attack from a MITM preimage attack.

In this chapter, we give a generic method to convert a particular preimage attack into a collision attack. By using our technique, an efficient collision attack which works faster than a generic collision attack can be constructed from a partial target preimage attack even if the complexity of the preimage attack is more than the birthday bound ($2^{n/2}$). Our method is especially fit for converting a MITM preimage attack into a pseudo collision attack, since most of the recent MITM preimage attacks can be considered as the partial target preimage attack as long as its matching point is located in the end of the compression function. We first apply our algorithm to SHA-256 and SHA-512 and show the best pseudo collision attacks on them in literature. Specifically, pseudo collisions of the 43-step (out of 64-step) reduced SHA-256 and the 46-step (out of 80-step) reduced SHA-512 can be derived faster than a generic attack. Combined with the recent preimage attacks on SHA-2 [73], these attacks are extended to the 52-step and 57-step reduced SHA-256 and SHA-512, respectively. Then we show some other applications of our conversion techniques including a pseudo collision attack on the 37-round reduced Skein-512. While it seems hard to extend our pseudo collision attacks to collision attacks, the proposed conversion technique is a generic, and thus it is expected to be widely used for security evaluations of hash functions.

11.2 Preliminaries

In this section, we first give security notions used throughout this chapter, then briefly refer a meet-in-the-middle (MITM) preimage attack.

11.2.1 Security Notions

Let f be a compression function which outputs an n-bit chaining variable h_i from an n-bit input chaining variable h_{i-1} and a k-bit input message m_i, i.e., $h_i = f(h_{i-1}, m_i)$. Similarly, let H be an iterated hash function consisting of f, which produces an n-bit hash value d from an initial value $IV(= h_0)$ and an arbitrary length message M, i.e., $d = H(IV, M) = f(\cdots f(f(IV, m_1), m_2), \cdots, m_t)$, where $pad(M) = (m_1|m_2|\cdots|m_t)$ and pad denotes a padding function. This type of hash function, in which the size of an intermediate chaining variable is the same as that of a hash value, is called a *narrow-pipe* hash function. On the other hand, a hash function having a larger internal state size is called a *wide-pipe* hash function, i.e., the size of a final hash value is smaller than that of a chaining variable. We use the terminology introduced in [141] for a collision attack and a pseudo (or free-start) collision attack on hash functions as follows.

Definition 1 (Collision attack) Given IV, find (M, M') such that $M \neq M'$ and $H(IV, M) = H(IV, M')$.

Definition 2 (Free-start or pseudo collision attack) Find (IV, IV', M, M') such that $H(IV, M) = H(IV', M')$ and $(IV, M) \neq (IV', M')$.

Additionally, we give several definitions for (pseudo) preimage attacks on hash functions and (pseudo) preimage attacks on compression functions.

Definition 3 (Preimage attack) Given IV and $d(= H(IV, M))$, find M' such that $H(IV,\ M') = d$.

Definition 4 (Pseudo preimage attack) Given $d(= H(IV, M))$, find (IV', M') such that $H(IV', M') = d$.

Definition 5 ((*t*-bit) partial target preimage attack) Given IV and t-bit partial target of $d(= H(IV, M))$, find M' such that t-bit of $d'(= H(IV, M'))$ is the same as the t-bit of d at the same position, and the other part of d' is randomly obtained.

Definition 6 (Preimage attack on compression function) Given h_{i-1} and $h_i(= f(h_{i-1},\ m_i))$, find m'_i such that $f(h_{i-1}, m'_i) = h_i$.

Definition 7 (Pseudo preimage attack on compression function) Given $h_i(= f(h_{i-1},\ m_i))$, find (h'_{i-1}, m'_i) such that $f(h'_{i-1}, m'_i) = h_i$.

Definition 8 ((*t*-bit) partial target preimage attack on compression function) Given h_{i-1} and t-bit partial target of $h_i(= f(h_{i-1}, m_i))$, find m'_i such that t-bit of $h'_i(= f(h_{i-1}, m'_i))$ is the same as the t-bit of h_i at the same position, and the other part of h'_i is randomly obtained.

Definition 9 ((*t*-bit) pseudo partial target preimage attack on compression function) Given t-bit partial target of $h_i(= f(h_{i-1}, m_i))$, find (h'_{i-1}, m'_i) such that t-bit of $h'_i(= f(h'_{i-1}, m'_i))$ is the same as the t-bit of h_i at the same position, and the other part of h'_i is randomly obtained.

11.2.2 Meet-in-the-Middle Preimage Attack

The basic concept of the MITM preimage attack was introduced in [16]. Since then, the MITM preimage attacks have been drastically improved and applied to several hash functions [142, 62, 82, 81, 84, 85]. Also, the techniques for the MITM preimage attacks on hash functions have been extended to the attacks on several block ciphers [9, 86].

As shown in Fig. 11.1,[1] in the MITM preimage attack on a compression function,

[1]Here, we show the MITM preimage attack on Davies-Meyer mode as an example. MITM

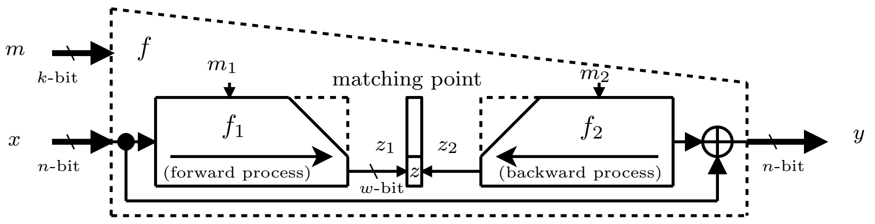


Figure 11.1: Meet-in-the-middle preimage attack

the compression function f is assumed to be divided into two sub-functions: f_1 (forward process) and f_2 (backward process) so that the w-bit matching point z calculated by f_1 does not depend on m_2 which is some message bits of m, and z calculated by f_2 does not depend on m_1 which is other message bits of m. Such m_1 and m_2 are called neutral bits of f_2 and f_1, respectively. Then, the MITM preimage attack finds a preimage m' such that $f(x, m') = y$ from a given x and $y (= f(x, m))$ as follows.

Step 1. Choose a random m except for m_1 and m_2.

Step 2. For all possible m_1, calculate w-bit $z_1 (= f_1(x, m_1))$, and add a pair of $(z_1^{(i)}, m_1^{(i)})$ to a list, where $(1 \le i \le 2^{|m_1|})$, and $| * |$ denotes the bit size of $*$.

Step 3. For all possible m_2, calculate w-bit $z_2 (= f_2^{-1}(x \oplus y, m_2))$, and add a pair of $(z_2^{(j)}, m_2^{(j)})$ to a list, where $(1 \le j \le 2^{|m_2|})$.

Step 4. Compare two lists to find pairs satisfying $z_1^{(p)} = z_2^{(q)}$. If such pair is found, then check if the other bits of the matching point derived from $m_1^{(p)}$ and $m_2^{(q)}$ are the same value.

Step 5. If the other parts are also the same, then outputs such m including $m_1^{(p)}$ and $m_2^{(q)}$. Otherwise, go back to Step 1 and repeat the computation.

From Steps 2 and 3, we have $2^{|m_1|}$ and $2^{|m_2|}$ values of w-bit z_1 and z_2, i.e., we have $2^{|m_1|+|m_2|}$ values of $(z_1 \oplus z_2)$. Since the probability of $(z_1 \oplus z_2 = 0)$ is 2^{-w}, we have $2^{|m_1|+|m_2|} \cdot 2^{-w}$ pairs such that $z_1 = z_2$ in Step 4. Thus, by repeating this algorithm about $2^{n-w} \cdot 2^{-(|m_1|+|m_2|)} \cdot 2^{w}$ times, we expect to obtain a desired preimage. The required computation for the one process from Step 1 to 4 is at most $\max(2^{|m_1|}, 2^{|m_2|})$ calls of the compression function. Thus, the total computation to find a preimage of the compression function is about $2^{n} \cdot 2^{-(|m_1|+|m_2|)} \cdot \max(2^{|m_1|}, 2^{|m_2|})$.[2]

For a narrow-pipe hash function, by replacing x and y by IV and d, this MITM preimage attack on a compression function can be directly converted into a preimage attack on a hash function. However, for an attack on a hash function, some of the

preimage attacks on other modes like Matyas-Meyer-Oseas mode can be performed in a similar way.

[2]The estimated complexity does not depend on the size of the matching point w. However, as discussed in [85], if w is extremely small like $w = 1$, the total complexity is dominated by the recomputations in Step 4 which is ignored in our estimation. Thus, in our evaluation, we assume that w is sufficiently large.

message bits related to the padding bits are required to be controlled by the attacker to set appropriate padding data.

11.3 Method to Convert Preimage Attack into Collision Attack

In this section, we present how to efficiently convert a particular preimage attack into a pseudo collision attack. First, we introduce a generic technique to construct a pseudo collision attack from a partial target preimage attack. Then, we introduce the MITM preimage attack whose matching point is located at the end of the compression function. We show that such class of the MITM preimage attack is regarded as the partial target preimage attack. Finally, we show that a pseudo collision attack can be efficiently constructed from the MITM preimage attack whose matching point is at the end by showing how to efficiently obtain many partial target preimages.

11.3.1 Generic Conversion of Partial Target Preimage Attack into Collision Attack

We consider the oracle $\mathcal{A}$ that can find a t-bit partial target preimage with a complexity of 2^s. Also, $\mathcal{A}$ is assumed to return different M' for each call. Obviously, we can construct a collision attack with a complexity of $2^s \cdot 2^{(n-t)/2}$ by iteratively calling $\mathcal{A}$ as follows.

- Set t-bit random data as d'
- Call $\mathcal{A}$ with the parameter IV and d' in $2^{(n-t)/2}$ times

After this procedure, we have $2^{(n-t)/2}$ of $(n-t)$-bit random data, and thus there exists a colliding data with a high probability. Once the colliding data are found, we have a collision of the hash function since the rest of the hash value d' is fixed. The total complexity is $2^{(n-t)/2} \cdot 2^s$. This conversion itself can be applied to not only a narrow-pipe hash function but also a wide-pipe hash function, since the required complexity depends only on the size of the digest. The basic concept of this attack that fixes t-bit of the target with the complexity of 2^s has been used to find a collision of (new) FORK-256 in [143] and a collision and a second preimage of LUX in [144]. However, the method does not work if the partial target preimage attack is not efficient, i.e., $(s \geq t/2)$. In this case, the required complexity in total will be higher than $2^{n/2}$.

11.3.2 Meet-in-the-Middle Attack with Matching Point in Last Step

We consider a similar model explained in Section 11.2.2. The difference from the model shown in Fig. 11.1 is that the matching point is restricted to be in the last step as shown in Fig. 11.2. In this scenario, the MITM pseudo preimage attack on a compression function finds a preimage m' and a random x' such that $f(x', m') = y$ from a given $y(= f(x, m))$ as follows.

Step 1. Choose a random m except for m_1 and m_2, and a random starting state S.

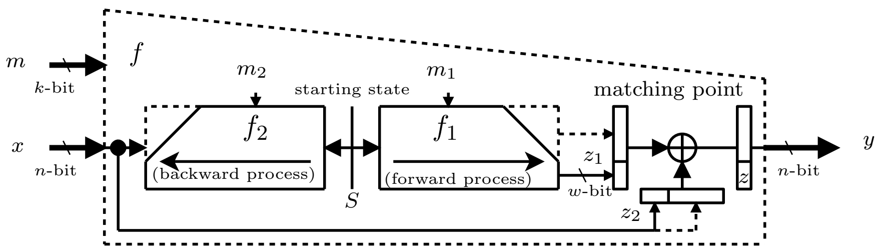


Figure 11.2: MITM preimage attack with the matching point in the last step

Step 2. For all possible m_1, calculate w-bit $z_1(= f_1(S, m_1))$, and add a pair of $(z_1^{(i)}, m_1^{(i)})$ to a list, where $(1 \leq i \leq 2^{|m_1|})$.

Step 3. For all possible m_2, calculate w-bit $z_2(= f_2^{-1}(S, m_2))$, and add a pair of $(z_2^{(j)}, m_2^{(j)})$ to a list, where $(1 \leq j \leq 2^{|m_2|})$.

Step 4. Compare two lists to find pairs satisfying that $z_1^{(p)} \oplus z_2^{(q)}$ equals the t-bit of y. If such pair is found, then check if the XORed other bits of the matching point derived from $m_1^{(p)}$ and $m_2^{(q)}$ is the same as the rest of y.

Step 5. If the XORed other bits are also the same as y, then output such m including $m_1^{(p)}$ and $m_2^{(q)}$, and x' calculated from the data of the matching point. Otherwise, go back to Step 1 and repeat the computation.

Note that, this attack basically cannot obtain a preimage from the given x unlike the attack described in Section 11.2.2, since x' will be randomly derived. Thus, this attack is considered as a pseudo preimage attack on a compression function. However, for a narrow-pipe hash, it has been known that a pseudo preimage attack on a compression function can be converted into a preimage attack on a hash function assuming that the attacker can set valid padding bits [85]. The estimated complexity to find a desired pseudo preimage is the same as that presented in Section 11.2.2, i.e., $2^n \cdot 2^{-(|m_1|+|m_2|)} \cdot \max(2^{|m_1|}, 2^{|m_2|})$.

11.3.3 Conversion of MITM Preimage Attack into Pseudo Collision Attack

If we can construct the MITM pseudo preimage attack whose matching point is located at the end of the compression function, we can control part of the output variables as explained in the previous subsection. In other words, the MITM pseudo preimage attack described in the previous subsection can be regarded as the pseudo partial target preimage attack on a compression function. For the MITM preimage attack, at least $2^{t/2}$ computations are required to derive a preimage of an t-bit partial target. Thus, the directly converted pseudo collision attack will at least have the complexity of $2^{(n-t)/2+t/2} = 2^{n/2}$, that is not an efficient pseudo collision attack.

In order to overcome this problem, we exploit extra freedom of a neutral word after finding a partial target preimage. For example, in the case of $t = 10$ and $|m_1| = |m_2| = 8$ $(> t/2)$, we can find $2^6(= 2^{8+8}/2^{10})$ 10-bit partial target preimages with the complexity of 2^8. It essentially means that a 10-bit partial target preimage is found with the complexity of $2^2 (= 2^8/2^6) < 2^5 (= 2^{10/2})$. When $t \leq w$, the required complexity to find a partial target preimage from a given t-bit partial target is estimated as

$$2^{t-(|m_1|+|m_2|)} \cdot \max(2^{|m_1|}, 2^{|m_2|}),$$

where recall that w denotes the bit size of the matching point. In particular, $s < t/2$, which is the condition for a successful attack as mentioned in Section 11.3.1, holds when $\min(|m_1|, |m_2|) > t/2$, where recall that 2^s represents the required complexity to find a t-bit partial target preimage. Therefore, if we can move the matching point of the MITM attack to the end of the compression function and there is enough freedom in neutral words, we can construct an efficient pseudo collision attack on a compression function.

Moreover, for a narrow-pipe hash function, it has been known that a (pseudo) collision attack on a compression function can be directly converted to a (pseudo) collision attack on a hash function by appending another message block illustrated in Fig. 11.3, which is called multi-block message technique. By using the multi-block message technique, an attacker can append arbitrary messages. Thus, unlike the conversion to a (pseudo) preimage attack on a hash function, for the conversion to a pseudo collision attack on a hash function, there is no restriction on controllability of message bits for a MITM pseudo preimage attack on a compression function. This will relax conditions on the position of the matching point for the MITM pseudo preimage attack on a compression function, and thus may allow us to attack larger number of steps. Note that, for a wide-pipe hash function, even though a (pseudo) collision attack on a compression function can not be directly converted to a (pseudo) collision attack on a hash function by using multi-block message, we still can convert a MITM pseudo preimage attack on a hash function to a pseudo collision attack on a hash function since the conversion of a partial target preimage attack into a collision attack is generic. Furthermore, in our attack, t-bit of the colliding digest can be determined by the attacker unlike the usual collision attack that derives a completely random digest. This is another feature of our approach. On the other hand, the required complexity of our converted (pseudo) collision attack is likely to be high due to a few gains from the MITM procedure, though it is still more efficient than the generic attack. This is considered as one of the limitations of our approach.

11.4 Pseudo Collision Attacks on SHA-2

In this section, we apply our conversion technique to SHA-2. At first, we briefly describe the algorithm of SHA-2. Then, we review the previous collision attacks on SHA-2. After that, we introduce the known MITM preimage attack on the 43-step SHA-256 presented in [84]. After we modify these results in order to fit our conversion technique, i.e., moving the matching point to the end of the compression function, we show the pseudo collision attack on the 43-step SHA-256. Moreover, we present the pseudo collision attack on the 46-step SHA-512 based on the MITM preimage attack on the 46-step SHA-512 [84]. Furthermore, pseudo collision attacks on the 40-step reduced SHA-224 and SHA-384 are demonstrated as well. Finally, we discuss pseudo collision attacks based on the recent MITM preimage attacks [73],

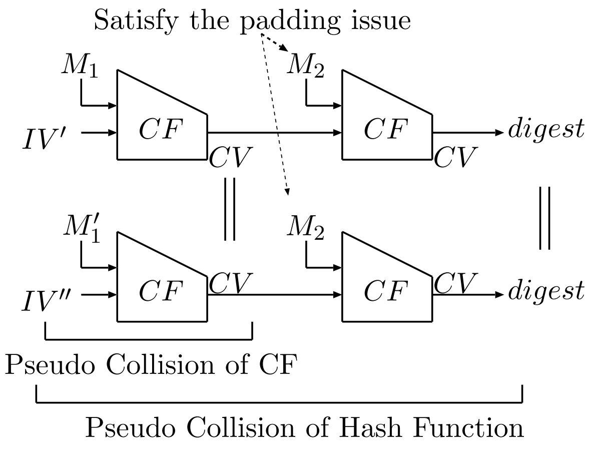


Figure 11.3: Multi-block pseudo collision

which significantly improve the results of [84] in terms of the number of attacked steps by using *bicliques*. These results on SHA-2 are summarized in Table 11.1.

11.4.1 Description of SHA-2

While our target is both SHA-256 and SHA-512, we only explain the structure of SHA-256, since SHA-512 is structurally equivalent to SHA-256 except for the number of steps, the amount of rotations and the word size. The compression function of SHA-256 consists of a message expansion function and a state update function. The message expansion function expands a 512-bit message block into 64 32-bit message words $(W_0, \cdots, W_{63})$ as follows:

$$W_i = \begin{cases} M_i & (0 \le i < 16), \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & (16 \le i < 64), \end{cases}$$

where the functions $\sigma_0(X)$ and $\sigma_1(X)$ are defined by

$$\begin{aligned} \sigma_0(X) &= (X \ggg 7) \oplus (X \ggg 18) \oplus (X \gg 3), \\ \sigma_1(X) &= (X \ggg 17) \oplus (X \ggg 19) \oplus (X \gg 10). \end{aligned}$$

The state update function updates eight 32-bit chaining variables, A, B, $\cdots$, G, H in 64 steps as follows:

$$\begin{aligned} T_1 &= H_i + \Sigma_1(E_i) + Ch(E_i, F_i, G_i) + K_i + W_i, \\ T_2 &= \Sigma_0(A_i) + Maj(A_i, B_i, C_i), \\ A_{i+1} &= T_1 + T_2,\ B_{i+1} = A_i,\ C_{i+1} = B_i,\ D_{i+1} = C_i, \\ E_{i+1} &= D_i + T_1,\ F_{i+1} = E_i,\ G_{i+1} = F_i,\ H_{i+1} = G_i, \end{aligned}$$

where K_i is the i-th step constant and the functions Ch, Maj, Σ_0 and Σ_1 are given as follows:

$$\begin{aligned} Ch(X, Y, Z) &= XY \oplus \overline{X}Z, \\ Maj(X, Y, Z) &= XY \oplus YZ \oplus XZ, \\ \Sigma_0(X) &= (X \ggg 2) \oplus (X \ggg 13) \oplus (X \ggg 22), \\ \Sigma_1(X) &= (X \ggg 6) \oplus (X \ggg 11) \oplus (X \ggg 25). \end{aligned}$$

Table 11.1: Summary of collision attacks on the reduced SHA-2

algorithm	type of attack	steps	complexity	based attack	paper
SHA-256	collision	24	$2^{28.5}$	-	[145]
	collision	27	(practical)	-	[139]
	semi-free-start-collision^{*1}	24	2^{17}	-	[145]
	semi-free-start-collision^{*1}	32	(practical)	-	[139]
	pseudo-near-collision	31	2^{32}	-	[145]
	pseudo collision	42	2^{123}	[84]	Our (Section 11.4.7)
	pseudo collision	43	2^{126}	[84]	Our (Section 11.4.4)
	pseudo collision	45	$2^{126.5}$	[73]	Our (Section 11.4.9)
	pseudo collision	52	$2^{127.5}$	[73]	Our (Section 11.4.9)
SHA-224	pseudo collision	40	2^{110}	[84]	Our (Section 11.4.8)
SHA-512	collision	24	$2^{28.5}$	-	[145]
	pseudo collision	42	2^{244}	[84]	Our (Section 11.4.7)
	pseudo collision	46	$2^{254.5}$	[84]	Our (Section 11.4.6)
	pseudo collision	50	$2^{254.5}$	[73]	Our (Section 11.4.9)
	pseudo collision	57	$2^{255.5}$	[73]	Our (Section 11.4.9)
SHA-384	pseudo collision	40	2^{183}	[84]	Our (Section 11.4.8)

$*1$: semi-free-start-collision attack finds (IV', M, M') such that $H(IV', M) = H(IV', M')$ and $M \neq M'$.

After 64 steps, a feed-forward process is executed with initial state variables by using word-wise addition modulo 2^{32}.

11.4.2 Known Collision Attacks on SHA-2

The first collision attack on reduced SHA-256 was presented in [146] which is a 19-step near collision attack. Since then, the collision attacks on SHA-2 have been improved [147, 145, 139]. The previously published best collision attacks in terms of the number of attacked steps are the 27 steps on SHA-256 [139] and the 24 steps on SHA-512 [145]. A non-random property, which is a second-order differential collision, of the 47-step reduced SHA-256 compression function was reported in [148].

11.4.3 Known MITM Preimage Attack on 43-step SHA-256

The MITM preimage attack on the 43-step SHA-256 presented in [84] uses the 33-step two chunks $W_j, \ldots, W_{j+32}$ including the 4-step initial structure (IS), the 2-step partial fixing (PF), the 7-step partial matching (PM) and the 1-step indirect partial matching (IPM). In the following, we review the details of these techniques.

33-step Two Chunks with the 4-step IS.

The message words of length 33 is divided into two chunks as $\{W_j, \ldots, W_{j+14}, W_{j+18}\}$ and $\{W_{j+15}$, W_{j+16}, W_{j+17}, $W_{j+19}, \ldots, W_{j+32}\}$. Using message compensation technique [84], the first chunk and the second chunk are independent from W_{j+15} and W_{j+18}, respectively. In particular, the following constraints ensure the above message words to be neutral words with respect to each chunk;

$$\begin{array}{llllll} W_{j+17} &= \sigma_1(W_{j+15}), & W_{j+19} &= \sigma_1^2(W_{j+15}), & W_{j+21} &= \sigma_1^3(W_{j+15}), \\ W_{j+22} &= W_{z+5}, & W_{j+23} &= \sigma_1^4(W_{j+15}), & W_{j+24} &= 2\sigma_1(W_{j+15}), \\ W_{j+25} &= \sigma_1^5(W_{j+15}), & & & & \end{array} \quad (11.1)$$

where $\sigma_1^2(X)$ means $\sigma_1 \circ \sigma_1(X)$.

These two chunks include the 4-step IS, which essentially exchanges the order of the words W_i and W_{i+3} by exploiting the absorption property of the function Ch. After the swapping, the final output after the step $(i+3)$ still keeps unchanged. Here, W_{j+18} is moved to the first chunk and W_{j+15}, W_{j+16} and W_{j+17} are moved to the second chunk.

In the forward direction, a state value of $p_{j+33} = A_{j+33}||\ldots||H_{j+33}$ can be computed independently of the first chunk. In the backward direction, a state value of $p_j = A_j||\ldots||H_j$ can be computed independently of the second chunk. Note that the 33-step two-chunk is valid regardless of the choice of j for $j > 0$.

7-step PM.

In the backward computation, A_j can be computed from p_{j+7} without knowing $\{W_j, \cdots, W_{j+6}\}$ for any j as used in [81].

2-step PF.

PF is a technique to enhance PM by fixing a part of a neutral word. The equation for H_{j-1} is as follows:

$$\begin{cases} H_{j-1} = & A_j - \Sigma_0(B_j) - Maj(B_j, C_j, D_j) - \Sigma_1(F_j) \\ & -Ch(F_j, G_j, H_j) - K_{j-1} - W_{j-1}, \\ W_{j-1} = & W_{j+15} - \sigma_1(W_{j+13}) - W_{j+8} + \sigma_0(W_j). \end{cases}$$

If we fix the lower ℓ bits of W_{j+15}, which is assumed to be a neutral word for the other chunk, the lower ℓ bits of H_{j-1} can be computed without using the value of the higher $(32-\ell)$ bits of W_{j+15}. Furthermore, the equation for H_{j-2} is expressed as follows:

$$\begin{cases} H_{j-2} = & A_{j-1} - \Sigma_0(B_{j-1}) - Maj(B_{j-1}, C_{j-1}, D_{j-1}) - \Sigma_1(F_{j-1}) \\ & -Ch(F_{j-1}, G_{j-1}, H_{j-1}) - K_{j-2} - W_{j-2}, \\ W_{j-2} = & W_{j+14} - \sigma_1(W_{j+12}) - W_{j+7} + \sigma_0(W_{j-1}). \end{cases}$$

The lower $(\ell - 18)$ bits of H_{j-2} can be computed if we can obtain the lower ℓ bits of $Ch(F_{j-1}, G_{j-1}, H_{j-1})$ and the lower $(\ell-18)$ bits of $\sigma_0(W_{j-1})$. Note that these values can be computed by using only the lower ℓ bits of W_{j+15}. Thus, when we fix the lower ℓ bits of W_{j+15}, the lower $(\ell - 18)$ bits of H_{j-2} can be computed without knowing the higher $(32-\ell)$ bits of W_{j+15}. Therefore, by combining the 7-step PM with the 2-step PF, 9 steps can be skipped in the backward computation.

1-step IPM.

For the forward computation, A_{j+34} can be expressed as a sum of two independent functions ψ_F, ξ_F of each neutral word as follows;

$$\begin{cases} A_{j+34} = & \Sigma_0(A_{j+33}) + Maj(A_{j+33}, B_{j+33}, C_{j+33}) + H_{j+33} + \Sigma_1(A_{j+33}) \\ & +Ch(A_{j+33}, B_{j+33}, C_{j+33}) + K_{j+33} + W_{j+33}, \\ W_{j+33} = & \sigma_1(W_{j+31}) + W_{j+26} + \sigma_0(W_{j+18}) + W_{j+17}, \end{cases}$$
$$\Rightarrow A_{j+34} = \psi_F(W_{j+15}) + \xi_F(W_{j+18}).$$

Then, we can compute $\psi_F(W_{j+15})$ and $\xi_F(W_{j+18})$ independently. It is equivalent to move the computation of $\xi_F(W_{j+18})$ to the backward chunk. In this case, $\xi_F(W_{j+18}) = \sigma_0(W_{j+18})$.

Attack Overview.

These techniques enable us to construct the 43 (= 33 + 7 + 2 + 1)-step attack on SHA-256. Here, we have the freedom of choice of j as long as 36 steps (W_{j-2} to W_{j+34}) is located sequentially.

For the actual attack in [84], j is chosen as $j = 3$, because W_{13}, W_{14} and W_{15} can be freely chosen to satisfy the message padding rule. The matching state is the lower 4 bits of A_{37}. In addition, the number of fixed bits ℓ for PF is chosen as $\ell = 23$. Then, neutral words of W_{18} and W_{21} have 5- and 4-bit freedom degrees, respectively. As a result, a pseudo preimage is found with the complexity of $2^{251.9}$. After that, pseudo preimages are converted into a preimage with the complexity of $2^{254.9}$. See [84] for more details about this attack.

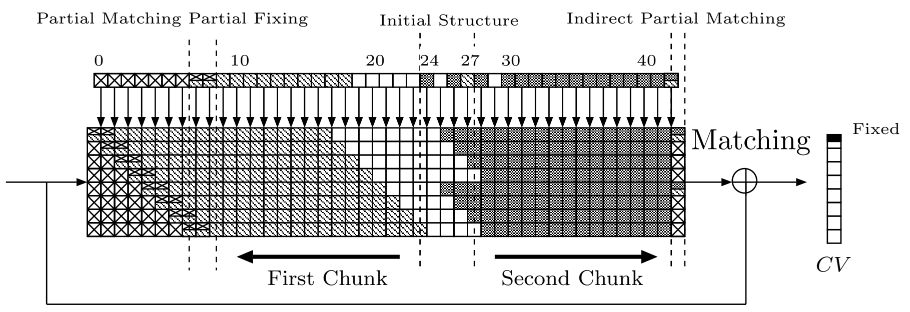
11.4.4 Pseudo Collision Attack on 43-step SHA-256

As discussed in Section 11.3.3, to convert a MITM preimage attack into a pseudo collision attack, the matching point is located into the end of the compression function, i.e., the addition of the feed-forward. As mentioned in section 11.4.3, the matching point of the 43-step MITM preimage attack is selected at the state after the step 37 ($j = 3$) due to the padding bits.

However, for a (pseudo) collision attack, we do not need to control message words for satisfying the padding rules, since we can generate correct padding by simply adding another message block as discussed in Section 11.3.3. It means that the last block of a compression function is used only for satisfying the padding condition in the collision attack when pseudo collision can be found before the last compression function as shown in Fig. 11.3. As a result, for a (pseudo) collision attack, we can move the matching point to the state after the step 43 ($j = 9$) that is the end of the compression function. [3]

Let a 256-bit output of the compression function be $CV = \{Z_A|| \cdots ||Z_H\}$, where each word is 32 bits. For $j = 9$, W_{24} and W_{27} are neutral words, and the matching point is the lower 4 bits of $A_{43} (= A_0 \oplus Z_A)$.

In order to construct the pseudo collision attack, we give the efficient method to obtain 4-bit partial target preimages by using the MITM technique [84]. Figure 11.4 shows the overview of the 43-step pseudo collision attack.



▧ : variables not depending on W_{24} ⊠ : variables depending on both W_{24} and W_{27}

▩ : variables not depending on W_{27} ▦ : a few bits of variables depending only on W_{27}

▨ : variables that can be expressed as a sum module 2^{32} of W_{24} and W_{27}

Figure 11.4: 43-step pseudo collision attack on SHA-256

Attack Procedure.

1. Choose the lower 4 bits of Z_A, which are target values.
2. Randomly choose the value of p_{25} and message W_{25}. Randomly fix the lower 23 bits of W_{24}. Then we can find 2^5 values of W_{24} on average from 9 free bits that correctly construct the 4-step initial structure and store them in the table T_W.
3. Randomly choose message words not related to the initial structure and the neutral words, i.e., W_{19}, W_{20}, W_{21}, W_{22}, W_{23} and W_{29} (called an initial configuration).
4. For all 2^5 possible W_{24} in T_W, compute W_{26}, W_{28}, W_{30}, W_{31}, W_{32}, W_{33} and W_{34} following Eq. (11.1). Compute forward and find $\psi_F(W_{24})$. Then, store the pairs (W_{24}, $\psi_F(W_{24})$) in a list L_F.
5. For all 2^4 possible values (the lower 4 bits) of W_{27}, compute backward and find $\xi_F(W_{27})$ and the lower 4 bits of A_0. Then, store the pairs $(W_{27}, Z_A \oplus A_0 - \sigma_0(W_{27}))$ in a list L_B.
6. If a match is found, i.e., $\psi_F(W_{24}) = Z_A \oplus A_0 - \sigma_0(W_{27})$, then compute two group of states $A_{43}, B_{43}, \cdots, H_{43}$ and $A_0, B_0, \cdots, H_0$ with corresponding W_{24} and W_{27}, respectively. Then obtain 2^5 $(= 2^9/2^4)$ CV whose 4-bit are fixed, i.e., the lower 4 bits of Z_A, and store these in a List L_1.
7. Repeat (3)-(6) 2^{121} times with different values of the initial configuration.

After the above procedures, we obtain 2^{126} $(= 2^5 \times 2^{121})$ pairs whose 4 bits are fixed.[4] Thus, there exists a colliding pair with a high probability, because of the

[3]It is also pointed out in [85] as the matching point can be rotated to the end of the compression function

[4]It is noted that we need a slightly more than 2^{121} times repeated experiments to get 2^{126} pairs that will achieve a probability higher than 2^{-1}. However the difference is so small that we ignore it here.

equation of ($2^{126} = 2^{(256-4)/2}$).

Evaluation.

We assume that the complexity for the 1-step function and the 1-step message expansion is 1/43 compression function operation of the 43-step SHA-256. As estimated in [85], the complexity of Step 2 in the presented attack is 2^9, and that of Steps 3-6 is $2^{4.878}$, which is the complexity for finding 2^5 4-bit partial target preimages. Thus, whole complexity of the pseudo collision attack on the 43-step SHA-256 is estimated as $2^{126} \approx 2^9 + (2^{121} \times 2^{4.878})$.

11.4.5 Known MITM Preimage Attack on 46-step SHA-512

The MITM preimage attack on the 46-step SHA-512 presented in [84] uses the 31-step two chunks $W_j, \dots, W_{j+30}$ including the 2-step IS, the 8-step PF for $W_{j-1}, \dots, W_{j-6}$ and W_{j+31},W_{j+32} and the 7-step PM. In this attack, we can choose j as long as 39 step (W_{j-6} to W_{j+32}) are located sequentially. For the actual attack in [85], j is chosen as $j = 6$ to satisfy the padding rule. Then, the neutral words W_{21} and W_{22} have 4 and 3-bit freedom degrees, respectively, and the bit size of the matching point is 3. Thus, a preimage of the 46-step SHA-512 is found with the complexity of $2^{511.5}$. See [84] for more details about this attack.

11.4.6 Pseudo Collision Attack on 46-step SHA-512

Similarly to the attack on the reduced SHA-256, we can move the matching point to the end of the compression function, because the padding issue can be avoided by using multi-block message technique in the pseudo collision attack. In the case of SHA-512, since the bit size of the matching point is 3, we utilize the 3-bit partial target preimages for the attack. Then, the complexity of the attack is estimated as $2^{254.5} = (2^{(512-3)/2})$.

11.4.7 Pseudo Collision Attacks on 42-step SHA-256 and 42-step SHA-512

We consider pseudo collision attacks on smaller number of rounds of SHA-2 in order to save the time complexity. For the 42-step reduced SHA-256, we can use 10 bits of freedom in both directions to find a 10-bit partial target preimage as discussed in Section 5.4 of [84]. This implies that a 10-bit partial target preimage is obtained with the complexity 1 ($< 2^5$). Thus, a pseudo collision is found with the complexity of $2^{123}(= 2^{(256-10)/2} \times 2^{10}/2^{10})$. Similarly to this, for the 42-step reduced SHA-512, we can use 24 bits of freedom in both directions to find a 24-bit partial target preimage as discussed in Section 6.5 of [84]. Therefore, a pseudo collision of the 42-step reduced SHA-512 is found with the complexity of $2^{244}(= 2^{(512-24)/2} \times 2^{24}/2^{24})$.

11.4.8 Pseudo Collision Attacks on Reduced SHA-224 and SHA-384

The pseudo collision attack on the 43-step SHA-256 described in Section 11.4.4 is applicable to the 43-step SHA-224 in the similar manner. However, we can not use the multi-block message technique straightforwardly, because the pseudo collision attack on SHA-224 needs to be done in the last compression function whose output Z_H is disregarded. Thus, due to the padding issue, we can mount only pseudo collision attack on a compression function of 43-step, not a hash function. The estimated complexity is 2^{110} for this attack.

However, the smaller number of rounds of SHA-224 hash function can be attacked by using another MITM attack. The 40-step SHA-224 hash function can be attacked by using the same two chunks for the 43-step preimage attack on SHA-256 in [84], i.e., the case of $j = 3$. The 7-step partial matching for backward computation are replaced by the 4-step one. Then the message words W_{13}, W_{14} and W_{15} are left as free message words to satisfy the padding rule. Instead of the lower 4 bits of Z_A, we use the lower 4 bits of Z_D as the target value. Here, we need additional one step: when finding matches at the lower 4 bits of A_{37}, we compute forward from the matching point to the end of the compression function (40-th step) by using these values that are computed forward from the starting point. Since $A_{37} = D_{40} = D_0 \oplus Z_D$ for the 40-step SHA-224, the lower 4 bits of Z_D will keep unaffected by the additional step. Thus, we can still get a partial target preimage. It can be converted into a pseudo collision attack on a hash function, because we can set W_{13}, W_{14} and W_{15} to follow the padding rule.

The detail of the attack procedure is as follows.

1. Choose the lower 4 bits of Z_D, which are target values.

2. Randomly choose the value of p_{19} and message W_{19}. Randomly fix the lower 23 bits of W_{18}. Then we can find 2^5 values of W_{18} on average from 9 free bits that correctly construct the 4-step initial structure and store them in the table T_W.

3. Randomly choose message words not related to the initial structure and the neutral words, i.e., W_{13}, W_{14}, W_{15}, W_{16}, W_{17}, W_{23} (called an initial configuration [84]).

4. For all 2^5 possible W_{18} in T_W, compute W_{20}, W_{22}, W_{24}, W_{25}, W_{26}, W_{27}, W_{28} following Eq, (11.1). Compute forward and find $\psi_F(W_{18})$. Store the pairs $(W_{18}, \psi_F(W_{18}))$ in a list L_F.

5. For all 2^4 possible values (the lower 4 bits) of W_{21}, compute backward and find $\xi_F(W_{21})$ and the lower 4 bits of A_{37} ($= D_{40} = Z_D \oplus D_0$). Store the pairs $(W_{21}, Z_D \oplus D_0 - \sigma_0(W_{27}))$ in a list L_B.

6. If a match is found, i.e., $\psi_F(W_{24}) = Z_D \oplus D_0 - \sigma_0(W_{27})$, then compute forward to get the states $A_{40}, B_{40}, \cdots, H_{40}$ with corresponding W_{24} and W_{27}, respectively. D_{40} will keep unaffected in this step. Then obtain 2^5 ($= 2^9/2^4$) CV whose 4 bits are fixed, i.e., the lower 4 bits of Z_D, and store these in a List.

7. Repeat (3)-(6) 2^{105} times with different values of the initial configuration.

The complexity of the attack is estimated as 2^{110}.

Similarly, the pseudo collision attack on the 46-step SHA-512 hash function described in 11.4.6 can also be applied to the 46-step SHA-384 compression function with the complexity of $2^{190.5} = (2^{(384-3)/2})$. For a pseudo collision attack on the reduced SHA-384 hash function, we use the 43-step preimage attack on SHA-384 [84]. Combining the result in [84] with our conversion technique, a pseudo collision attack on the 40-step SHA-384 hash function can be constructed. The matching bit is 18 when chosen parameter of partial matching as $\ell = 27$. The complexity of the pseudo collision attack on the 40-step SHA-384 is estimated as $2^{(384-18)/2} = 2^{183}$. These 40-step pseudo collision attacks give examples that the matching point is not at but near the end of compression function. That is compatible to solve padding problem.

11.4.9 Application to Other Results of SHA-2

Recently, the MITM preimage attacks on the reduced SHA-2 are improved by using "bicliques" technique which is considered as generalized initial structure [73]. This technique enables us to construct longer initial structures than those of the attacks [84]. In the following, let us consider pseudo collision attacks based on [73].

For SHA-256, the 36-step two independent chunks including the 6-step IS based on bicliques are constructed. Combining the 2-step PM with the 7-step PM and the 1-step IPM, the MITM preimage attack on the 45-step SHA-2 is derived. In this attack, both neutral words have 3-bit freedom degrees, and the matching point is 4-bit. Since our conversion technique does not need to consider the padding issue, the matching point can be moved to the end of the compression function similar to the 43-step attack. Then, we can convert it into the 45-step pseudo collision attack on SHA-256 with the complexity of $2^{126.5}$ $(= 2^{(256-3)/2})$ [5]. Similarly, we can construct the 50-step pseudo collision attack on SHA-512 based on the 50-step MITM preimage attack [73]. In this attack, both neutral words have 3-bit freedom degrees, and the bit size of the matching point is 3. Thus, the complexity of the attack is estimated as $2^{254.5}$ $(= 2^{(512-3)/2})$.

In addition, [73] showed pseudo preimage attacks on the 52-step SHA-256 and the 57-step SHA-512. For the setting of a pseudo preimage attack, the cost of converting a pseudo preimage to a preimage is omitted. Thus, larger number of rounds can be attacked. Note that in these attacks, the amount of freedom degrees for both neutral words are only 1-bit, and the bit size of the matching point is 1. In order to construct a pseudo collision attack by using our conversion technique, it is sufficient to obtain a pseudo preimage on a compression function, i.e., a preimage on a hash function is not needed. Therefore, the above explained pseudo preimage attacks can also be converted into pseudo collision attacks in a similar way. The complexities of the pseudo collision attacks on the 52-step SHA-256 and the 57-step SHA-512 are estimated as $2^{127.5}$ $(= 2^{(256-1)/2})$ and $2^{255.5}$ $(= 2^{(512-1)/2})$, respectively.

11.5 Application to Skein

In this section, we show pseudo collision attacks on the reduced Skein-512 [41] based on the preimage attacks presented in [73].

[5]Our attack uses only 3 bits for the matching and find 3-bit partial target preimages, because this setting is optimal with respect to the time complexity.

Table 11.2: Parameters of the (pseudo) preimage attacks on the reduced Skein-512

Parameters of the preimage attack on the 22-round Skein-512 hash function

Chunks			Matching			
Forward	Backward	Biclique	Partial matching	Matching bits	Total matching pairs	Complexity
8-11	16-19	12-15	$20 \to 24 = 3 \leftarrow 7$	$I^1_{30,31,53}$	2^3	$2^{2.3}$

Parameters of the pseudo preimage attack on the 37-round Skein-512 compression function

Chunks			Matching			
Forward	Backward	Biclique	Partial matching	Matching bits	Total matching pairs	Complexity
8-15	24-31	16-23	$32 \to 38 = 2 \leftarrow 7$	I^3_{25}	2	$2^{1.2}$

11.5.1 Description of Skein

Skein is built from the tweakable block cipher Threefish $E_{K,T}(P)$, where K, T and P denote a key, a tweak and a plaintext message, respectively. The compression function $F(CV, T, M)$ of Skein outputs the next chaining variable as $F(CV, T, M) = E_{CV,T}(M) \oplus M$, where CV is the previous chaining variable and M is an input message block.

Threefish-512 supports a 512-bit block and a 512-bit key, and operates on 64-bit words. The subkey $K^s = (K^s_0, K^s_1, \ldots, K^s_7)$ injected every four rounds is generated from the secret key $K = K[0], K[1], \ldots, K[7]$ as follows:

$$\begin{array}{rclrcl} K^s_j & = & K[(s+j) \bmod 9], (0 \leq j \leq 4); & K^s_5 & = & K[(s+5) \bmod 9] + T[s \bmod 3]; \\ K^s_6 & = & K[(s+6) \bmod 9] + T[(s+1) \bmod 3]; & K^s_7 & = & K[(s+7) \bmod 9] + s, \end{array}$$

where s denotes a round counter, $T[0]$ and $T[1]$ denote tweak words, $T[2] = T[0] + T[1]$, and $K[8] = C_{240} \oplus \bigoplus_{j=0}^{7} K[j]$ with a constant C_{240}. Each Threefish-512 round consists of four MIX functions followed by a permutation of the eight 64-bit words. The 128-bit function MIX processes the pairs of eight words of internal state $I^0, I^1, \ldots, I^7$ after key addition.

11.5.2 Known Pseudo Preimage Attacks on Skein

We briefly review two MITM preimage attacks on Skein-512 presented in [73]: one is a preimage attack on the 22-round reduced Skein-512 hash function starting from the 3rd round, and the other is a preimage attack on the 37-round reduced Skein-512 compression function starting from the 2nd round.

For the 22-round attack, the 3-dimension biclique at rounds 12-15 is obtained with the complexity of 2^{200}. Since many bicliques can be produced out of one, the cost of constructing the bicliques is negligible in the total complexity of the attack. In this attack, we can obtain 2^3 pairs matched in 3 bits by $2^{2.3}$ calls of the 22-round Skein-512 compression function. As a result, a preimage of the 22-round reduced Skein is found with the complexity of $2^{511.2}$.

Considering a pseudo preimage attack on the compression function, it is natural to assume that tweak bits T can also be controlled by the attacker. Due to additional freedom, the pseudo preimage attack on the 37-round reduced Skein-512 is feasible by using the 1-dimension biclique at rounds 16-23. In this attack, we can obtain 2 pairs matched in 1 bit by $2^{1.2}$ calls of the 37-round Skein-512 compression function.

Consequently, a pseudo preimage of the 37-round reduced Skein is found with the complexity of $2^{511.2}$.

The parameters for the preimage attacks on the 22-round and the 37-round reduced Skein-512 hash function and compression function are summarized in Table 11.2. See [73] for more details about this attack.

11.5.3 Pseudo Collision Attacks on Skein.

Since the matching point used in the MITM preimage attack on the 22-round reduced Skein-512 hash function [73] is located in the end of the compression function, our conversion technique can directly convert it to the pseudo collision attack on the 22-round reduced Skein-512. In this attack, the neutral words have 3-bit freedom degrees, and the bit size of the matching point is 3. As reported in [73], a 3-bit matching candidate can be found with the complexity of $2^{2.3}/2^{3}$. Thus, the complexity of the pseudo collision attack on the 22-round reduced Skein-512 hash function is estimated as $2^{253.8}$ $(= 2^{(512-3)/2} \times 2^{2.3}/2^{3})$.

The pseudo preimage attack on the 37-round reduced Skein compression function can be converted into a pseudo collision attack on a hash function in a similar way. The required complexity for the pseudo collision attack on the 37-round reduced Skein hash function is estimated as $2^{255.7}$ $(= 2^{(512-1)/2} \times 2^{1.2}/2)$.

11.6 Conclusion

In this chapter, we gave a generic method to convert preimage attacks to pseudo collision attacks. It provides a new insight to evaluate the security of hash functions. The essence of the method is converting a partial target preimage attack to a pseudo collision attack. That is especially compatible to meet-in-the-middle preimage attacks since it can be converted into a partial target preimage attack if the matching point can be moved to the end of a hash function or a compression function and enough freedom on neutral bits are left.

Using the proposed approach, we presented the best pseudo collision attacks on SHA-2 based on the known preimage attacks, which has been left as open question. We showed pseudo collision attacks on the 43- and 46-step reduced SHA-256 and SHA-512 based on the MITM preimage attacks presented in [84]. Also, pseudo collision attacks on the 52- and 57-step reduced SHA-256 and SHA-512 based on the more advanced MITM preimage attacks in [73] were demonstrated. We also applied the conversion technique to other hash functions including Skein with the meet-in-the-middle preimage attacks, which showed the widely usage of this method. The pseudo collision attacks on the 22- and 37-round reduced Skein-512 were presented. Our technique may also apply to other hash functions, such as Tiger [38]. Based on the MITM preimage attack on the full Tiger [85], we might construct the pseudo collision attack on the full Tiger. We believe that the technique can be used for more hash algorithms once their preimage or pseudo preimage attacks are found.

By this method, now we only can get pseudo collision attacks. It is left as future works that how to construct collision attacks from known preimage attacks.

Part IV

Design of Block Cipher

Chapter 12

A Block Cipher *Piccolo*

We gives several feedbacks for secure designs of block ciphers, stream ciphers and hash functions. Then, we introduce a 64-bit blockcipher *Piccolo* supporting 80 and 128-bit keys as a secure block cipher. Adopting secure design criteria, *Piccolo* achieves high security. We show that *Piccolo* offers a sufficient security level against known analyses including recent related-key differential attacks and meet-in-the-middle attacks.

12.1 Feedback for Secure Design

12.1.1 Feedback for Block Cipher

- Self similarities among round functions should be avoided. The property causes several weaknesses such as slide attacks [10, 65], a reflection property [18], a reflection meet-in-the-middle attack (Chapter 3) and equivalent keys on the short number of rounds (Chapter 3). To destroy self similarities, round constants should be used.
- Lightweight block ciphers often adopt the simple key scheduling functions in order to to reduce the gate size. Since the MITM attack mainly exploits a low key-dependency, the MITM attack becomes the most powerful attack for such ciphers with respect to the number of attack rounds. If the related key setting is allowed, the low key-dependency property is also exploited for a boomerang-type attacks [8, 27]. Actually, in the single key setting, the MITM attack is best attacks on Piccolo, LED, XTEA (Chapter 4) and KATAN 48/64 (Chapter 5). In the related key setting, the related-key boomerang attack is best attacks on KATAN 32/48/64 (Chapter 6). When designing lightweight block ciphers, the number of round should be considered by carefully analyzing the immunity of MITM attacks and related-key differential-type attacks.
- For MITM attacks, the diffusion property of key information in the data processing part is also important. If the large parts of the cipher are independent from some key bits, it may be vulnerable to MITM attacks. In addition, the internal state should be nonlinearly affected by a key, because linear relations between the key and the state are disregarded by the indirect matching technique [84]. Thus, a key bits should be nonlinearly-diffused in the data

processing part as fast as possible. Also, all the key bits should be used in a small number of rounds.

- The all subkey recovery approach assumes that a key scheduling function is an ideal function (Chapter 6). Thus, it gives generic lower bounds of the security of several construction against the MITM attacks, regardless of the construction of key scheduling functions.

12.1.2 Feedback for Stream Cipher

- Different pairs of keys and IVs must generate different keystreams. The existence of such pairs means that the effective (K, IV) space is smaller than the expected one which is the sum of the K and IV size. Also, the IV collision may leak the information of the internal state (Chapter 7 and [31]). Thus the key scheduling algorithm should be an injective function with respect to a key and an IV.
- It should get rid of the sliding property such that two different Key-IV pairs generate same keystream but n-bit shifted. In the related key setting, a keystream is easily distinguished from a random keystream, and a key may be efficiently recovered (Chapter 8). To destroy self similarities, using round constants in the key scheduling function is recommended.
- If the internal-state size is much larger than the key size, the sufficient randomization is required when converting the key information into the internal state in the the key scheduling algorithm. If it is not sufficient, bytes (or bits) of the internal-state tend to correlate each other. It may lead to attacks based on the internal-state correlation such as guess and determine attacks (Chapter 7) and efficient state-recovery attacks (Chapter 9 and [128]).
- Keystreams should not be biased to particular values. It allows to mount not only distinguishing attacks, but also plaintext recovery attacks in the ciphertext only setting (Chapter 9). The sufficient randomizatin of the internal state before outputting a keystream are required.

12.1.3 Feedback for Hash Function

- The MITM preimage attack exploits the low diffusion property and the simple message scheduling function (Chapter 10). Countermeasures of this attack are almost same as those of the MITM attack on block ciphers. However, the purpose of the MITM preimage attack of hash functions is to efficiently find the internal state by controlling message values, while that of block ciphers is to detect wrong keys using the internal state relation. Thus, the size of internal state depend on the time complexity of preimage attacks unlike attacks on block ciphers. Therefore, the larger state is one of meaningful countermeasures.
- New relation between a meet-in-the-middle preimage attack and a collision attack is found (Chapter 11). Thus, even if a hash function requires only a collision resistance, i.e., the preimage resistance is not needed, the MITM attack should be taken into consideration.

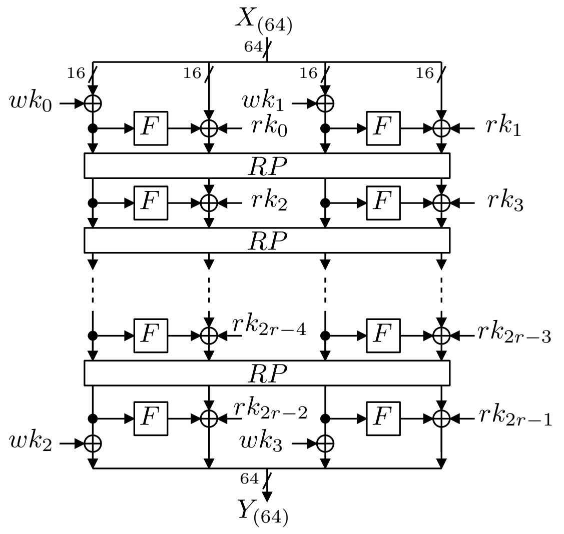
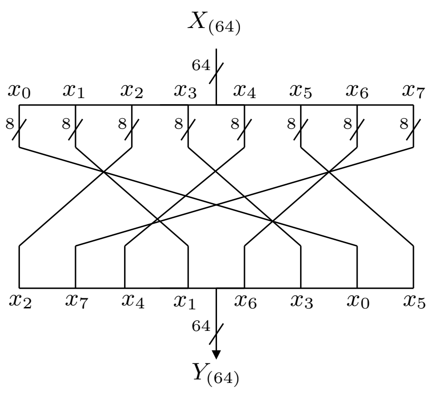


Figure 12.1: Encryption function G_r



$X_{(64)}$
64
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
8
x_2 x_7 x_4 x_1 x_6 x_3 x_0 x_5
$Y_{(64)}$

Figure 12.2: Round permutation *RP*

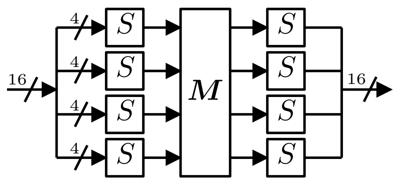


Figure 12.3: F-function

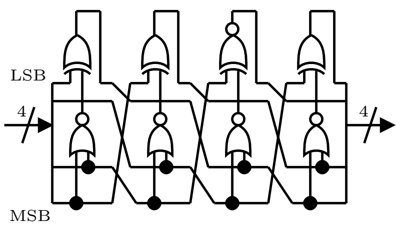


Figure 12.4: S-box

12.2 Specification of *Piccolo*

This section provides the specification of *Piccolo*. *Piccolo* is a 64-bit blockcipher supporting 80 and 128-bit keys. The 80 and the 128-bit key mode are referred as *Piccolo*-80 and *Piccolo*-128, respectively. Both ciphers consist of a data processing part and a key scheduling part. The differences between two key modes lie in the number of rounds for the data processing part and the key scheduling part. We first give notations used throughout this chapter, then define each part.

12.2.1 Notations

$a_{(b)}$	:	b denotes the bit length of a.
$a\|b$ or $(a\|b)$	:	Concatenation.
$a \leftarrow b$	:	Updating a value of a by a value of b.
${}^t\boldsymbol{a}$	:	Transposition of a vector or a matrix $\boldsymbol{a}$.
$\{a\}_b$	:	Representation in base b.

12.2.2 Data Processing Part

The data processing part of *Piccolo* consisting of r rounds, G_r, takes a 64-bit data $X \in \{0,1\}^{64}$, four 16-bit whitening keys $wk_i \in \{0,1\}^{16} (0 \le i < 4)$ and $2r$ 16-bit round keys $rk_i \in \{0,1\}^{16} (0 \le i < 2r)$ as the inputs, and outputs a 64-bit data

$Y \in \{0,1\}^{64}$. G_r is defined as follows:

$$G_r : \left\{ \begin{array}{l} \{0,1\}^{64} \times \{\{0,1\}^{16}\}^4 \times \{\{0,1\}^{16}\}^{2r} \rightarrow \{0,1\}^{64} \\ (X_{(64)}, wk_{0(16)}, ..., wk_{3(16)}, rk_{0(16)}, ..., rk_{2r-1(16)}) \mapsto Y_{(64)} \end{array} \right.$$

```
**Algorithm** $G_r(X_{(64)}, wk_0, ..., wk_3, rk_0, ..., rk_{2r-1})$ **:**
$X_{0(16)}|X_{1(16)}|X_{2(16)}|X_{3(16)} \leftarrow X_{(64)}$
$X_0 \leftarrow X_0 \oplus wk_0$, $X_2 \leftarrow X_2 \oplus wk_1$
for $i \leftarrow 0$ to $r-2$ do
    $X_1 \leftarrow X_1 \oplus F(X_0) \oplus rk_{2i}$, $X_3 \leftarrow X_3 \oplus F(X_2) \oplus rk_{2i+1}$
    $X_0|X_1|X_2|X_3 \leftarrow RP(X_0|X_1|X_2|X_3)$
$X_1 \leftarrow X_1 \oplus F(X_0) \oplus rk_{2r-2}$, $X_3 \leftarrow X_3 \oplus F(X_2) \oplus rk_{2r-1}$
$X_0 \leftarrow X_0 \oplus wk_2$, $X_2 \leftarrow X_2 \oplus wk_3$
$Y_{(64)} \leftarrow X_0|X_1|X_2|X_3$
```

where F is a 16-bit F-function and RP is a 64-bit permutation defined in the following sections. The decryption function G_r^{-1} is obtained from G_r by simply changing the order of whitening and round keys as follows:

$$G_r^{-1} : \left\{ \begin{array}{l} \{0,1\}^{64} \times \{\{0,1\}^{16}\}^4 \times \{\{0,1\}^{16}\}^{2r} \rightarrow \{0,1\}^{64} \\ (Y_{(64)}, wk_{0(16)}, ..., wk_{3(16)}, rk_{0(16)}, ..., rk_{2r-1(16)}) \mapsto X_{(64)} \end{array} \right.$$

```
**Algorithm** $G_r^{-1}(Y_{(64)}, wk_0, ..., wk_3, rk_0, ..., rk_{2r-1})$ **:**
$wk'_0 \leftarrow wk_2, wk'_1 \leftarrow wk_3, wk'_2 \leftarrow wk_0, wk'_3 \leftarrow wk_1$
for $i \leftarrow 0$ to $r-1$ do
    $rk'_{2i}|rk'_{2i+1} \leftarrow \left\{ \begin{array}{ll} rk_{2r-2i-2}|rk_{2r-2i-1} & \text{(if } i \bmod 2 = 0\text{)} \\ rk_{2r-2i-1}|rk_{2r-2i-2} & \text{(if } i \bmod 2 = 1\text{)} \end{array} \right.$
$X_{(64)} \leftarrow G_r(Y, wk'_0, ..., wk'_3, rk'_0, ..., rk'_{2r-1})$
```

The number of rounds, r, is 25 and 31 for *Piccolo*-80 and -128, i.e., G_{25} and G_{31} for *Piccolo*-80 and -128, respectively (See Fig. 12.1).

F-Function.

F-function $F : \{0,1\}^{16} \rightarrow \{0,1\}^{16}$ consists of two S-box layers separated by a diffusion matrix (See Fig. 12.3). The S-box layer consists of four 4-bit bijective S-boxes S given by Table 12.1, and updates a 16-bit data $X_{(16)}$ as follows:

$$(x_{0(4)}, x_{1(4)}, x_{2(4)}, x_{3(4)}) \leftarrow (S(x_{0(4)}), S(x_{1(4)}), S(x_{2(4)}), S(x_{3(4)})),$$

where $X_{(16)} = x_{0(4)}|x_{1(4)}|x_{2(4)}|x_{3(4)}$. The diffusion matrix $\boldsymbol{M}$ is defined as

$$\boldsymbol{M} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}.$$

Then the diffusion function updates a 16-bit data $X_{(16)}$ as follows:

$${}^t(x_{0(4)}, x_{1(4)}, x_{2(4)}, x_{3(4)}) \leftarrow \boldsymbol{M} \cdot {}^t(x_{0(4)}, x_{1(4)}, x_{2(4)}, x_{3(4)}),$$

where the multiplications between matrices and vectors are performed over GF(2^4) defined by an irreducible polynomial $x^4 + x + 1$.

Table 12.1: 4-bit bijective S-box S in hexadecimal form

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

Round Permutation.

The round permutation $RP : \{0,1\}^{64} \to \{0,1\}^{64}$ divides a 64-bit input $X_{(64)}$ into eight 8-bit data as $X_{(64)} = x_{0(8)}|x_{1(8)}|...|x_{7(8)}$, then permutes them by the following manner:

$$RP : (x_{0(8)}, x_{1(8)}, ..., x_{7(8)}) \leftarrow (x_{2(8)}, x_{7(8)}, x_{4(8)}, x_{1(8)}, x_{6(8)}, x_{3(8)}, x_{0(8)}, x_{5(8)}).$$

Finally, the round permutation concatenates $(x_{0(8)}, x_{1(8)}, ..., x_{7(8)})$ into $X_{(64)}$ (See Fig. 12.2).

12.2.3 Key Scheduling Part

The key scheduling part of *Piccolo* supports 80 and 128-bit keys and outputs 16-bit whitening keys $wk_{i(16)}(0 \le i < 4)$ and round keys $rk_{j(16)}(0 \le j < 2r)$ for the data processing part. The key scheduling functions for *Piccolo*-80 and -128 are referred as KS_r^{80} and KS_r^{128}, respectively. We first define 16-bit constants con_i^{80} and con_i^{128}, then describe each key schedule.

Constant Values.

The constants con_i^{80} and con_i^{128} used in KS_r^{80} and KS_r^{128}, respectively, are generated as follows:

$$\begin{cases} (con_{2i}^{80}|con_{2i+1}^{80}) & \leftarrow & (c_{i+1}|c_0|c_{i+1}|\{00\}_2|c_{i+1}|c_0|c_{i+1}) \oplus \{\texttt{0f1e2d3c}\}_{16}, \\ (con_{2i}^{128}|con_{2i+1}^{128}) & \leftarrow & (c_{i+1}|c_0|c_{i+1}|\{00\}_2|c_{i+1}|c_0|c_{i+1}) \oplus \{\texttt{6547a98b}\}_{16}, \end{cases}$$

where c_i is a 5-bit representation of i, e.g., $c_{11} = \{\texttt{01011}\}_2$.

Key Schedule for 80-Bit Key Mode (KS_r^{80}).

The key scheduling function for the 80-bit key mode, KS_r^{80}, divides an 80-bit key $K_{(80)}$ into five 16-bit sub-keys $k_{i(16)}$ $(0 \le i < 5)$ and provides $wk_{i(16)}(0 \le i < 4)$ and $rk_{j(16)}(0 \le j < 2r)$ as follows:

```
Algorithm $KS_r^{80}(K_{(80)})$ :
$wk_0 \leftarrow k_0^L|k_1^R, wk_1 \leftarrow k_1^L|k_0^R, wk_2 \leftarrow k_4^L|k_3^R, wk_3 \leftarrow k_3^L|k_4^R$
for $i \leftarrow 0$ to $(r-1)$ do
    $(rk_{2i}, rk_{2i+1}) \leftarrow (con_{2i}^{80}, con_{2i+1}^{80}) \oplus \begin{cases} (k_2, k_3) & \text{(if } i \bmod 5 = 0 \text{ or } 2) \\ (k_0, k_1) & \text{(if } i \bmod 5 = 1 \text{ or } 4) \\ (k_4, k_4) & \text{(if } i \bmod 5 = 3), \end{cases}$
```

where k_i^L and k_i^R are left and right half 8 bits of k_i, respectively, i.e., $k_{i(16)} = k_{i(8)}^L|k_{i(8)}^R$ and $k_{i(8)}^R$ contains the least significant bit of $k_{i(16)}$.

Table 12.2: Min. # differentially and linearly active F-functions (single-key setting)

rounds	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
min. # active F-functions	0	1	2	3	4	6	7	8	9	10	11	12	13	14	15
rounds	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
min. # active F-functions	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Key Schedule for 128-Bit Key Mode (KS_r^{128}).

The key scheduling function for the 128-bit key mode, KS_r^{128}, divides a 128-bit key $K_{(128)}$ into eight 16-bit sub-keys $k_{i(16)}$ $(0 \leq i < 8)$ and provides $wk_{i(16)}(0 \leq i < 4)$ and $rk_{j(16)}(0 \leq j < 2r)$ as follows:

```
**Algorithm** $KS_r^{128}(K_{(128)})$ :
$wk_0 \leftarrow k_0^L|k_1^R$, $wk_1 \leftarrow k_1^L|k_0^R$, $wk_2 \leftarrow k_4^L|k_7^R$, $wk_3 \leftarrow k_7^L|k_4^R$
for $i \leftarrow 0$ to $(2r-1)$ do
    if $(i+2)$ mod $8 = 0$ then
        $(k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7) \leftarrow (k_2, k_1, k_6, k_7, k_0, k_3, k_4, k_5)$
    $rk_i \leftarrow k_{(i+2) \bmod 8} \oplus con_i^{128}$
```

12.3 Security Evaluation of Piccolo

In this section, we provide results on security analysis for *Piccolo*. We show that *Piccolo* offers a sufficient security level against known analyses including recent related-key differential attacks and meet-in-the-middle attacks.

12.3.1 Differential Attack / Linear Attack

We first show the minimum numbers of differentially and linearly active F-functions of G_r up to 30 rounds in Table 12.2. Note that the minimum numbers for differentially and linearly active F-functions are the same due to the duality of differential and linear attacks and the similarity of G_r and G_r^{-1}. MDP and MLP of the F-function are $2^{-9.3}$ and $2^{-8.0}$, respectively. Combining those results, *Piccolo* consisting of at least 7 or 8 rounds provide at least 7 or 8 active F-functions, and have no differential or linear trails whose probabilities are more than 2^{-64}, respectively. Thus, we expect that the full-round of *Piccolo* (25 and 31 rounds for *Piccolo*-80 and -128) has enough immunity against differential and linear attacks, since it has large security margin.

12.3.2 Boomerang-Type Attacks

The boomerang-type attacks (including the boomerang, amplified boomerang and rectangle attacks) first divide the cipher into two sub-ciphers, then find a boomerang quartet with high probability. The probability of constructing a boomerang quartet is denoted as $\hat{p}^2\hat{q}^2$, where $\hat{p} = \sqrt{\sum_\beta \Pr^2[\alpha \to \beta]}$, and α and β are input and output differences for the first sub-cipher, and $\hat{q}$ for the second sub-cipher. $\hat{p}^2$ is bounded by the maximum differential trail probability, i.e., $\hat{p}^2 \leq \max_\beta \Pr[\alpha \to \beta]$, and $\hat{q}^2$

as well. Let p, q be the maximum differential trail probability for the first and the second sub-ciphers. Then, p, q are bounded by multiplying the minimum number of active F-functions in each sub-cipher with MDP of the F-function. From Table 12.2, any combination of two sub-ciphers for *Piccolo* consisting of at least 9 rounds has at least 7 active F-functions in total. Hence, we conclude that the full-round of *Piccolo* is sufficiently secure against boomerang-type attacks.

12.3.3 Impossible Differential Attack

An impossible differential attack is likely to be applied to a variant of GFN due to its slow diffusion. However, *Piccolo* utilizes the round permutation RP to achieve faster diffusion compared to a standard type-II GFN. Then, for both encryption and decryption sides, *Piccolo* requires only four rounds to be full diffusion, which is a property that all outputs are affected by all inputs. We also search the longest impossible differential by modified $\mathcal{U}$-method algorithm and found a 7-round impossible differential exploiting a 4-bit truncated differential. Therefore, we conclude that the full-round of *Piccolo* is expected to be secure against the impossible differential attack.

12.3.4 Related-Key Differential Attacks

In the related-key setting, a distinguisher is allowed to use related-keys and usually uses key differentials to cancel out differentials in a data processing part. While the practical impact of related-key differential attacks is still controversial, we care about it from a pessimistic (designers') point of view. We evaluate the immunity against related-key differential attacks by counting the minimum number of differentially active F-functions in the related-key setting. Table 12.3 shows the minimum numbers of differentially active F-functions for the 80 and the 128-bit key modes up to 20 rounds. Unlike the attacks under the single-key setting, the total number of active F-functions for the related-key differential attacks may vary according to the starting round. However, in our evaluations, those differences are at most 2 active F-functions, even if the starting round is changed. Consequently, we obtain that over 14 and 16 rounds for *Piccolo*-80 and -128 have at least 7 differentially active F-functions in the related-key setting, respectively.

Moreover, we consider related-key boomerang/rectangle attacks [27]. Similarly to non related-key boomerang-type attacks, we evaluate the security in the worst case that an attacker can use pq instead of $\hat{p}^2\hat{q}^2$ for the probability of a boomerang quartet. As a result, we confirmed that over 17 and 21 rounds of *Piccolo*-80 and -128 provide enough (seven) differentially active F-functions in this setting.

Furthermore, we take related-key impossible differential attacks [154] into account. Consequently, by using modified $\mathcal{U}$-method, we found an 11 and a 17-round impossible differential distinguisher using an 8-bit truncated differential for *Piccolo*-80 and -128 in the related-key setting, respectively, and they are the longest in our evaluation. Therefore, we conclude that the full-round *Piccolo* is expected to be resistant to those attacks.

Table 12.3: Min. # differentially active F-functions (related-key setting)

rounds / starting round i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
for *Piccolo*-80 encryption																				
$i \bmod 5 = 0$	0	0	0	0	0	2	3	4	4	5	5	6	7	7	7	8	9	10	11	11
$i \bmod 5 = 1$	0	0	0	0	1	2	3	4	4	5	6	6	7	7	8	9	9	10	11	11
$i \bmod 5 = 2$	0	0	0	0	1	2	3	3	4	6	6	6	7	7	9	9	9	10	10	12
$i \bmod 5 = 3$	0	0	0	0	1	2	2	3	4	5	5	6	6	7	8	8	9	9	10	11
$i \bmod 5 = 4$	0	0	0	0	0	0	2	3	4	5	6	6	7	7	7	7	9	10	11	11
for *Piccolo*-80 decryption																				
$i \bmod 5 = 0$	0	0	0	0	1	2	2	3	4	5	5	6	6	7	8	8	9	9	10	11
$i \bmod 5 = 1$	0	0	0	0	1	2	3	3	4	6	6	6	7	7	9	9	9	10	10	12
$i \bmod 5 = 2$	0	0	0	0	1	2	3	4	4	5	6	6	7	7	8	9	9	10	11	11
$i \bmod 5 = 3$	0	0	0	0	0	2	3	4	4	5	5	6	7	7	7	8	9	10	11	11
$i \bmod 5 = 4$	0	0	0	0	0	0	2	3	4	5	6	6	7	7	7	7	9	10	11	11
for *Piccolo*-128 encryption																				
$i \bmod 4 = 0$	0	0	0	0	0	0	0	1	3	3	4	5	5	6	7	7	8	9	10	10
$i \bmod 4 = 1$	0	0	0	0	0	0	1	2	3	3	4	5	5	6	7	7	8	9	10	11
$i \bmod 4 = 2$	0	0	0	0	0	0	1	2	2	3	4	4	5	6	6	7	7	9	9	9
$i \bmod 4 = 3$	0	0	0	0	0	1	1	1	2	3	4	5	5	6	7	7	8	9	9	10
for *Piccolo*-128 decryption																				
$i \bmod 4 = 0$	0	0	0	0	0	1	1	2	3	3	4	5	5	6	6	7	8	9	9	11
$i \bmod 4 = 1$	0	0	0	0	0	0	1	2	3	3	4	4	5	6	7	7	8	9	9	9
$i \bmod 4 = 2$	0	0	0	0	0	0	0	1	2	3	4	5	5	6	7	7	7	9	10	10
$i \bmod 4 = 3$	0	0	0	0	0	0	1	1	2	3	4	5	5	6	7	7	8	9	10	10

12.3.5 Meet-in-the-Middle Attack

Three-subset meet-in-the-middle (MITM) cryptanalysis [9] is a recent attack on blockciphers. This attack works well for blockciphers having a simple key schedule and slow diffusion. Indeed, KTANTAN and GOST have been theoretically broken by this attack [9, 86]. Since *Piccolo* consists of the permutation based key scheduling and a variant of GFN, evaluating the resistance against this attack is important.

Similarly to data difference, *Piccolo* requires 4 rounds to non-linearly diffuse any round-key difference to all output data in the data processing part, i.e., any round-key bits of the i-th round non-linearly affect all input of the $(i-3)$-th round and all output of the $(i+3)$-th round. Thus, we assume that an attacker might construct an 8-round *indirect-partial matching* [82] and a 4-round *initial structure* [83] in the worst case. Besides, we even allow the attacker to use *code book* and *splice and cut* techniques [16]. In this worst setting, *Piccolo*-80 and -128 without whitening keys have neutral words up to 19 and 23 consecutive rounds, respectively. We expect that the attacked rounds obtained by this observation are upper bounds on the security against the three-subset MITM attack, since the given assumptions are sufficiently strong. Moreover, we attempt to construct actual attacks to obtain the lower bounds on the security. As a result, the *Piccolo*-80 and -128 without whitening keys reduced to 14 and 21 rounds can be attacked by the three-subset MITM attacks, respectively. Since *Piccolo* actually has whitening keys, it is obviously stronger than the variants evaluated above. Thus, we conclude that *Piccolo* has enough immunity against the three-subset MITM attack.

12.3.6 Other Attacks.

We also consider other attacks including a slide, a saturation, an interpolation, a higher order differential, a truncated differential, and an algebraic attack. Though the details of the evaluations for those attacks are omitted due to the page limitation, consequently, we expect that none of them work better than the previously explained attacks.

Chapter 13

Conclusion

This thesis has studied analysis and design of symmetric cryptographic algorithms. Especially, we focused on block ciphers, stream cipher and hash functions.

Analysis of Block cipher

We have improved the Meet-in-the-Middle (MITM) attack. At first, we developed the new variant of the MITM attack called Reflection-Meet-in-the-Middle (R-MITM) attack, and constructed the $first$ single-key attack on the full-round GOST block cipher.

We applied the MITM attack to lightweight block ciphers Piccolo, LED and XTEA, and updated best attacks of these ciphers with respect to the number of attack rounds in the single key setting.

We extended the MITM attack to apply it to wider class block ciphers, and proposed the new technique called all subkey recovery approach. It allows us to construct the MITM attack without dealing with the key scheduling function. Then we updated the best single-key attack on SHACAL-2, CAST, KATAN, Blowfish and FOX with respect to the number of attack rounds.

Also, we constructed related-key boomerang attacks, which relies on statistical weaknesses, on KATAN 32, 48 and 64, and improve best attacks.

Analysis of Stream Cipher

We have analyzed the security of several stream ciphers Py, RAKAPOSHI and RC4.

We showed the practical key recovery attack of Py by using the guess-and-determine approach in conjunction with the IV collision property.

We pointed out that RAKAPOSHI has the slide property, and showed that this property is exploitable for the key recovery attacks on RAKAPOSHI in the related-key setting.

we have introduced new biases in initial bytes of the keystrem of RC4, and demonstrated the plaintext recovery attack using these biases in the broadcast setting where the same plaintext is encrypted with different user keys.

Analysis of Hash Function

We have improved on the MITM technique for hash functions, and evaluated the security of well known hash function SHA-2, Tiger and Skein.

We proposed preimage attacks on reduced Tiger and SHA-2 by using the new technique, called independent transform, for finding neutral words for the MITM technique.

We provided the new insight of the relation between the meet-in-the-middle preimage attack and the pseudo collision attack, and presents a generic technique to convert a MITM preimage attacks into pseudo collision attacks. By using our technique, we updated best attack of pseudo collision attacks of SHA-256, SHA-512 and Skein with respect to the number of attack rounds.

Design of Block Cipher

We have given several feedbacks for secure designs of symmetric cryptographic algorithms. Then, we have introduced a 64-bit blockcipher Piccolo supporting 80 and 128- bit keys as a secure block cipher. Adopting secure design criteria, Piccolo achieves high security. We have shown that Piccolo offers a sufficient security level against known analyses including recent related-key differential attacks and meet-in-the-middle attacks.

Bibliography

[1] FIPS. Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, 2001. `http://www.cix.co.uk/klockstone/xtea.pdf`.

[2] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. The Keccak SHA-3 submission. Submission to NIST (Round 3), 2011.

[3] E. Biham and A. Shamir. Differential Cryptanalysis of DES-like Cryptosystems. In *CRYPTO*, pages 2–21, 1990.

[4] M. Matsui. Linear Cryptoanalysis Method for DES Cipher. In *EUROCRYPT*, pages 386–397, 1993.

[5] L. R. Knudsen. Truncated and Higher Order Differentials. In Preneel [155], pages 196–211.

[6] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. *J. Cryptology*, 7(4):229–246, 1994.

[7] E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In *EUROCRYPT*, pages 12–23, 1999.

[8] D. Wagner. The Boomerang Attack. In Knudsen [156], pages 156–170.

[9] A. Bogdanov and C. Rechberger. A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In Biryukov et al. [157], pages 229–240.

[10] A. Biryukov and D. Wagner. Slide Attacks. In Knudsen [156], pages 245–259.

[11] S. Lucks. The Saturation Attack - A Bait for Twofish. In Matsui [158], pages 1–15.

[12] X. Wang, H. Yu, and Y. L. Yin. Efficient Collision Search Attacks on SHA-0. In *CRYPTO*, pages 1–16, 2005.

[13] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In *EUROCRYPT*, pages 19–35, 2005.

[14] F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen. Rebound Attacks on the Reduced Grøstl Hash Function. In *CT-RSA*, pages 350–365, 2010.

[15] D. Khovratovich and I. Nikolic. Rotational Cryptanalysis of ARX. In *FSE*, pages 333–346, 2010.

[16] K. Aoki and Y. Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In Avanzi et al. [159], pages 103–119.

[17] National Soviet Bureau of Standards. Information Processing System - Cryptographic Protection - Cryptographic Algorithm GOST 28147-89, 1989.

[18] O. Kara. Reflection Cryptanalysis of Some Ciphers. In *INDOCRYPT*, pages 294–307, 2008.

[19] R. M. Needham and D. J. Wheeler. Tea extensions. Techniacl report, Computer Laboratory, University of Cambridge, 1997. `http://www.cix.co.uk/klockstone/xtea.pdf`.

[20] J. Guo, T. Peyrin, A. Poschmann, and M. J. B. Robshaw. The LED Block Cipher. In *CHES*, pages 326–341, 2011.

[21] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo: An Ultra-Lightweight Blockcipher. In *CHES*, pages 342–357, 2011.

[22] C. Adams. The CAST-128 encryption algorithm. RFC-2144, 1997.

[23] H. Handschuh and D. Naccache. SHACAL. NESSIE Proposal, 2001. `https://www.cosic.esat.kuleuven.be/nessie/updatedPhase2Specs/SHACAL/shacal-tweak.zip`.

[24] C. De. Cannière, O. Dunkelman, and M. Knezevic. KATAN and KTANTAN - A Family of Small and Efficient Hardware-Oriented Block Ciphers. In C. Clavier and K. Gaj, editors, *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[25] B. Schneier. Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). In *FSE*, pages 191–204, 1993.

[26] P. Junod and S. Vaudenay. FOX : A New Family of Block Ciphers. In *Selected Areas in Cryptography*, pages 114–129, 2004.

[27] E. Biham, O. Dunkelman, and N. Keller. Related-Key Boomerang and Rectangle Attacks. In *EUROCRYPT*, pages 507–525, 2005.

[28] E. Biham and J. Seberry. Py(Roo): a fast and secure stream cipher using rollingarrays. *eSTREAM*, Report 2005/023, 2005, 2005. `http://www.ecrypt.eu.org/stream/ciphers/py/py.ps`.

[29] E. Biham and J. Seberry. Another version of Py, 2005. `http://www.ecrypt.eu.org/stream/p2ciphers/py/pypy_p2.ps`.

[30] H. Wu and B. Preneel. Attacking the IV setup of Py and Pypy. *eSTREAM*, Report 2006/050, 2006, 2006. `http://www.ecrypt.eu.org/stream/papersdir/2006/050.pdf`.

[31] H. Wu and B. Preneel. Differential Cryptanalysis of the Stream Ciphers Py, Py6 and Pypy. In *EUROCRYPT*, pages 276–290, 2007.

[32] C. Cid, S. Kiyomoto, and J. Kurihara. The RAKAPOSHI Stream Cipher. In *ICICS*, pages 32–46, 2009.

[33] I. Mantin and A. Shamir. A Practical Attack on Broadcast RC4. In Matsui [158], pages 152–164.

[34] S. Maitra, G. Paul, and S. Sengupta. Attack on Broadcast RC4 Revisited. In *FSE*, pages 199–217, 2011.

[35] S. S. Gupta, S. Maitra, G. Paul, and S. Sarkar. (Non-)Random Sequences from (Non-)Random Permutations - Analysis of RC4 stream cipher. Cryptology ePrint Archive, Report 2011/448, 2011. `http://eprint.iacr.org/`.

[36] S. Sen Gupta, S. Maitra, G. Paul, and S. Sarkar. Proof of Empirical RC4 Biases and New Key Correlations. In *Selected Areas in Cryptography*, pages 151–168, 2011.

[37] I. Mantin. Predicting and Distinguishing Attacks on RC4 Keystream Generator. In *EUROCRYPT*, pages 491–506, 2005.

[38] R. J. Anderson and E. Biham. TIGER: A Fast New Hash Function. In *FSE*, pages 89–97, 1996.

[39] FIPS. Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180-4.

[40] S. Indesteege and B. Preneel. Preimages for Reduced-Round Tiger. In *WEWoRC*, pages 90–99, 2007.

[41] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker. The Skein Hash Function Family. Submission to NIST (Round 3), 2010. Available at http://www.skein-hash.info/sites/default/files/skein1.3.pdf.

[42] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[43] B. Schneier. *Applied Cryptography (2nd ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[44] M-J. O. Saarinen. A Chosen Key Attack against the Secret S-boxes of GOST(unpublished manuscript), 1998.

[45] A. Poschmann, S. Ling, and H. Wang. 256 Bit Standardized Crypto for 650 GE - GOST Revisited. In *CHES*, pages 219–233, 2010.

[46] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[47] J. Kelsey, B. Schneier, and D. Wagner. Key-Schedule Cryptoanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES. In *CRYPTO*, pages 237–251, 1996.

[48] H. Seki and T. Kaneko. Differential Cryptanalysis of Reduced Rounds of GOST. In *SAC*, pages 315–323, 2000.

[49] Y. Ko, S. Hong, W. Lee, S. Lee, and J-S. Kang. Related Key Differential Attacks on 27 Rounds of XTEA and Full-Round GOST. In *FSE*, pages 299–316, 2004.

[50] E. Fleischmann, M. Gorski, J. Hüehne, and S. Lucks. Key Recovery Attack on full GOST. Block Cipher with Negligible Time and Memory. In *Western European Workshop on Research in Cryptology (WEWoRC)*, volume 6429 of *LNCS*. Springer, 2009.

[51] V. Rudskoy. On Zero Practical Significance of " Key recovery attack on full GOST block cipher with zero time and memory ". Cryptology ePrint Archive, Report 2010/111, 2010. `http://eprint.iacr.org/`.

[52] E. Biham, O. Dunkelman, and N. Keller. Improved Slide Attacks. In Biryukov [160], pages 153–166.

[53] O. Dunkelman, N. Keller, and A. Shamir. Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In *ASIACRYPT*, pages 158–176, 2010.

[54] M. Steil. 17 Mistakes Microsoft Made in the Xbox Security System, 2005.

[55] D.J. Wheeler and R.M. Needham. TEA, a Tiny Encryption Algorithm. In Preneel [155], pages 363–366.

[56] W. Diffie and M. E. Hellman. Exhaustive Cryptanalysis of the NBS Data Encryption Standard. *Computer*, 10:74–84, 1977.

[57] D. Chaum and J. Evertse. Crytanalysis of DES with a Reduced Number of Rounds: Sequences of Linear Factors in Block Ciphers. In H.C. Williams, editor, *CRYPTO*, volume 218 of *Lecture Notes in Computer Science*, pages 192–211. Springer, 1985.

[58] H. Demirci and A. A. Selçuk. A Meet-in-the-Middle Attack on 8-Round AES. In *FSE*, pages 116–126, 2008.

[59] H. Demirci, I. Taskin, M. Çoban, and A. Baysal. Improved Meet-in-the-Middle Attacks on AES. In *INDOCRYPT*, pages 144–156, 2009.

[60] O. Dunkelman, G. Sekar, and B. Preneel. Improved Meet-in-the-Middle Attacks on Reduced-Round DES. In *INDOCRYPT*, pages 86–100, 2007.

[61] S. Indesteege, N. Keller, O. Dunkelman, E. Biham, and B. Preneel. A Practical Attack on KeeLoq. In *EUROCRYPT*, pages 1–18, 2008.

[62] Y. Sasaki and K. Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In *EUROCRYPT*, pages 134–152, 2009.

[63] O. Kara and C. Manap. A New Class of Weak Keys for Blowfish. In Biryukov [160], pages 167–180.

[64] B. Schneier. Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). In *FSE*, pages 191–204, 1993.

[65] A. Biryukov and D. Wagner. Advanced Slide Attacks. In B. Preneel, editor, *EUROCRYPT*, volume 1807 of *Lecture Notes in Computer Science*, pages 589–606. Springer, 2000.

[66] F. Mendel, N. Pramstaller, and C. Rechberger. A (Second) Preimage Attack on the GOST Hash Function. In *FSE*, pages 224–234, 2008.

[67] F. Mendel, N. Pramstaller, C. Rechberger, M. Kontak, and J. Szmidt. Cryptanalysis of the GOST Hash Function. In *CRYPTO*, pages 162–178, 2008.

[68] 3rd Generation Partnership Project. Technical Specification Group Services and System Aspects, 3G Security, Specification of the 3GPP Confidentiality and Integrity Algorithms; Document 2: KASUMI Specification, V3.1.1.

[69] O. Dunkelman, N. Keller, and A. Shamir. A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In *CRYPTO*, pages 393–410, 2010.

[70] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique Cryptanalysis of the Full AES. In *ASIACRYPT*, pages 344–371, 2011.

[71] D. Khovratovich, G. Leurent, and C. Rechberger. Narrow-Bicliques: Cryptanalysis of Full IDEA. In *EUROCRYPT*, pages 392–410, 2012.

[72] P. C. van Oorschot and Michael J. Wiener. A Known Plaintext Attack on Two-Key Triple Encryption. In *EUROCRYPT*, pages 318–325, 1990.

[73] D. Khovratovich, C. Rechberger, and A. Savelieva. Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In *FSE*, pages 244–263, 2012.

[74] G. Sekar, N. Mouha, V. Velichkov, and B. Preneel. Meet-in-the-Middle Attacks on Reduced-Round XTEA. In *CT-RSA*, pages 250–267, 2011.

[75] J. Chen, M. Wang, and B. Preneel. Impossible Differential Cryptanalysis of the Lightweight Block Ciphers TEA, XTEA and HIGHT. In *AFRICACRYPT*, pages 117–137, 2012.

[76] L. Wei, C. Rechberger, J. Guo, H. Wu, H. Wang, and S. Ling. Improved Meet-in-the-Middle Cryptanalysis of KTANTAN (Poster). In *ACISP*, pages 433–438, 2011.

[77] Y. Sasaki. Meet-in-the-Middle Preimage Attacks on AES Hashing Modes and an Application to Whirlpool. In *FSE*, pages 378–396, 2011.

[78] D. Hong, B. Koo, and D. Kwon. Biclique Attack on the Full HIGHT. In *ICISC*, pages 365–374, 2011.

[79] E. Biham, O. Dunkelman, N. Keller, and A. Shamir. New Data-Efficient Attacks on Reduced-Round IDEA. Cryptology ePrint Archive, Report 2011/417, 2011. `http://eprint.iacr.org/`.

[80] Mark Dermot Ryan, Ben Smyth, and Guilin Wang, editors. *Information Security Practice and Experience - 8th International Conference, ISPEC 2012, Hangzhou, China, April 9-12, 2012. Proceedings*, volume 7232 of *Lecture Notes in Computer Science*. Springer, 2012.

[81] T. Isobe and K. Shibutani. Preimage Attacks on Reduced Tiger and SHA-2. In *FSE*, pages 139–155, 2009.

[82] K. Aoki and Y. Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In *CRYPTO*, pages 70–89, 2009.

[83] Y. Sasaki and K. Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In *EUROCRYPT*, pages 134–152, 2009.

[84] K. Aoki, J. Guo, K. Matusiewicz, Y. Sasaki, and L. Wang. Preimages for Step-Reduced SHA-2. In *ASIACRYPT*, pages 578–597, 2009.

[85] J. Guo, S. Ling, C. Rechberger, and H. Wang. Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In *ASIACRYPT*, pages 56–75, 2010.

[86] T. Isobe. A Single-Key Attack on the Full GOST Block Cipher. In *FSE*, pages 290–305, 2011.

[87] Y. Sasaki, L. Wang, Y. Sakai, K. Sakiyama, and K. Ohta. Three-Subset Meet-in-the-Middle Attack on Reduced XTEA. In *AFRICACRYPT*, pages 138–154, 2012.

[88] T. Isobe and K. Shibutani. Security Analysis of the Lightweight Block Ciphers XTEA, LED and Piccolo. In *ACISP*, pages 71–86, 2012.

[89] P. C. van Oorschot and M. J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology*, 12(1):1–28, 1999.

[90] B. Schneier and J. Kelsey. Unbalanced Feistel Networks and Block Cipher Design. In *FSE*, pages 121–144, 1996.

[91] S. Vaudenay. On the Weak Keys of Blowfish. In *FSE*, pages 27–32, 1996.

[92] O. Kara and C. Manap. A New Class of Weak Keys for Blowfish. In Biryukov [160], pages 167–180.

[93] M. Wang, X. Wang, and C. Hu. New Linear Cryptanalytic Results of Reduced-Round of CAST-128 and CAST-256. In Avanzi et al. [159], pages 429–441.

[94] M. Wang, X. Wang, K. Chow, and L. C. K. Hui. New Differential Cryptanalytic Results for Reduced-Round CAST-128. *IEICE Transactions*, 93-A(12):2744–2754, 2010.

[95] NESSIE consortium. NESSIE portfolio of recommended cryptographic primitives, 2003. `https://www.cosic.esat.kuleuven.be/nessie/deliverables/decision-final.pdf`.

[96] Y. Shin, J. Kim, G. Kim, S. Hong, and S. Lee. Differential-Linear Type Attacks on Reduced Rounds of SHACAL-2. In *ACISP*, pages 110–122, 2004.

[97] V. Rijmen. Cryptanalysis and Design of Iterated Block Ciphers. Doctoral Dissertation, 1997.

[98] S. Knellwolf, W. Meier, and M. Naya-Plasencia. Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In *ASIACRYPT*, pages 130–145, 2010.

[99] W. Wu, W. Zhang, and D. Feng. Integral Cryptanalysis of Reduced FOX Block Cipher. In *ICISC*, pages 229–241, 2005.

[100] M. Ågren. Some Instant- and Practical-Time Related-Key Attacks on KTANTAN32/48/64. In *Selected Areas in Cryptography*, pages 213–229, 2011.

[101] T. Isobe and K. Shibutani. All Subkeys Recovery Attack on Block Ciphers: Extending Meet-in-the-Middle Approach. In *Selected Areas in Cryptography*, pages 202–221, 2012.

[102] M. R. Albrecht and G. Leander. An All-In-One Approach to Differential Cryptanalysis for Small Block Ciphers. In *Selected Areas in Cryptography*, pages 1–15, 2012.

[103] S. Knellwolf, Wi. Meier, and M. Naya-Plasencia. Conditional Differential Cryptanalysis of Trivium and KATAN. In *Selected Areas in Cryptography*, pages 200–212, 2011.

[104] J. Kelsey, T. Kohno, and B. Schneier. Amplified Boomerang Attacks Against Reduced-Round MARS and Serpent. In *FSE*, pages 75–93, 2000.

[105] E. Biham, O. Dunkelman, and N. Keller. The Rectangle Attack - Rectangling the Serpent. In *EUROCRYPT*, pages 340–357, 2001.

[106] The eSTREAM Project. `http://www.ecrypt.eu.org/stream`.

[107] S. Paul and B. Preneel an G. Sekar. Distinguishing Attacks on the Stream Cipher Py. In *FSE*, pages 405–421, 2006.

[108] P. Crowley. Improved cryptanalysis of Py. *IACR Cryptology ePrint Archive*, 2006:30, 2006.

[109] H. Wu and B. Preneel. Key recovery attack on Py and Pypy with chosen IVs. *eSTREAM*, Report 2006/050, 2006, 2006. `http://www.ecrypt.eu.org/stream/papersdir/2006/052.pdf`.

[110] J-P. Aumasson, L. Henzen, W. Meier, and M. Naya-Plasencia. Quark: A Lightweight Hash. In *CHES*, pages 1–15, 2010.

[111] J. Guo, T. Peyrin, and A. Poschmann. The PHOTON Family of Lightweight Hash Functions. In *CRYPTO*, pages 222–239, 2011.

[112] A. Bogdanov, M. Knezevic, G. Leander, D. Toz, K. Varici, and I. Verbauwhede. SPONGENT: A Lightweight Hash Function. In *CHES*, pages 312–325, 2011.

[113] M. Hell, T. Johansson, A. Maximov, and W. Meier. The Grain Family of Stream Ciphers. In *The eSTREAM Finalists*, pages 179–190. 2008.

[114] C. D. Cannière and B. Preneel. Trivium. In *The eSTREAM Finalists*, pages 244–266. 2008.

[115] S. Babbage and M. Dodd. The MICKEY Stream Ciphers. In *The eSTREAM Finalists*, pages 191–209. 2008.

[116] F. Arnault and T. P. Berger. F-FCSR: Design of a New Class of Stream Ciphers. In *FSE*, pages 83–97, 2005.

[117] M. Hell and T. Johansson. Breaking the Stream Ciphers F-FCSR-H and F-FCSR-16 in Real Time. *J. Cryptology*, 24(3):427–445, 2011.

[118] I. Dinur, T. Güneysu, C. Paar, A. Shamir, and R. Zimmermann. An Experimentally Verified Attack on Full Grain-128 Using Dedicated Reconfigurable Hardware. In *ASIACRYPT*, pages 327–343, 2011.

[119] S. Kiyomoto, T. Tanaka, and K. Sakurai. K2: A Stream Cipher Algorithm using Dynamic Feedback Control. In *SECRYPT*, pages 204–213, 2007.

[120] ISO/IEC 18033-4. Amendment 1 - Information technology - security techniques - Encryption algorithms - Part 4: Stream ciphers," JTC 1/SC 27 (IT security tech.) , 2011. `http://www.iso.org`.

[121] Y. Lee, K. Jeong, J. Sung, and S. Hong. Related-Key Chosen IV Attacks on Grain-v1 and Grain-128. In *ACISP*, pages 321–335, 2008.

[122] C. De Cannière, Ö. Küçük, and B. Preneel. Analysis of Grain's Initialization Algorithm. In *AFRICACRYPT*, pages 276–289, 2008.

[123] J. D. Golic. Linear Statistical Weakness of Alleged RC4 Keystream Generator. In *EUROCRYPT*, pages 226–238, 1997.

[124] S. R. Fluhrer and D. A. McGrew. Statistical analysis of the alleged rc4 keystream generator. In *FSE*, pages 19–30, 2000.

[125] I. Mironov. (Not So) Random Shuffles of RC4. In *CRYPTO*, pages 304–319, 2002.

[126] S. Paul and B. Preneel. A New Weakness in the RC4 Keystream Generator and an Approach to Improve the Security of the Cipher. In *FSE*, pages 245–259, 2004.

[127] L. R. Knudsen, W. Meier, B. Preneel, V. Rijmen, and S. Verdoolaege. Analysis Methods for (Alleged) RC4. In *ASIACRYPT*, pages 327–341, 1998.

[128] A. Maximov and D. Khovratovich. New State Recovery Attack on RC4. In *CRYPTO*, pages 297–316, 2008.

[129] M. Matsui. Key Collisions of the RC4 Stream Cipher. In *FSE*, pages 38–50, 2009.

[130] G. Paul and S. Maitra. Permutation After RC4 Key Scheduling Reveals the Secret Key. In *Selected Areas in Cryptography*, pages 360–377, 2007.

[131] E. Biham and Y. Carmeli. Efficient Reconstruction of RC4 Keys from Internal States. In *FSE*, pages 270–288, 2008.

[132] I. Mantin. Analysis of the Stream Cipher RC4. Master's Thesis, The Weizmann Institute of Science, Israel, 2001. `http://www.wisdom.weizmann.ac.il/~itsik/RC4/rc4.html`.

[133] P. Sepehrdad, S. Vaudenay, and M. Vuagnoux. Discovery and Exploitation of New Biases in RC4. In Biryukov et al. [157], pages 74–91.

[134] A. Roos. A class of weak keys in the RC4 stream cipher, 1995. Two posts in sci.crypt.

[135] B. Canvel, Al. P. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password Interception in a SSL/TLS Channel. In *CRYPTO*, pages 583–599, 2003.

[136] J. Kelsey and S. Lucks. Collisions and Near-Collisions for Reduced-Round Tiger. In *FSE*, pages 111–125, 2006.

[137] F. Mendel, B. Preneel, V. Rijmen, H. Yoshida, and D. Watanabe. Update on Tiger. In *INDOCRYPT*, pages 63–79, 2006.

[138] F. Mendel and V. Rijmen. Cryptanalysis of the Tiger Hash Function. In *ASIACRYPT*, pages 536–550, 2007.

[139] F. Mendel, T. Nad, and M. Schläffer. Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In *ASIACRYPT*, pages 288–307, 2011.

[140] C. De Cannière and C. Rechberger. Preimages for Reduced SHA-0 and SHA-1. In *CRYPTO*, pages 179–202, 2008.

[141] X. Lai and J. L. Massey. Hash Function Based on Block Ciphers. In *EUROCRYPT*, pages 55–70, 1992.

[142] Y. Sasaki and K. Aoki. Preimage attacks on 3, 4, and 5-pass haval. In *ASIACRYPT*, pages 253–271, 2008.

[143] M-J O. Saarinen. A Meet-in-the-Middle Collision Attack Against the New FORK-256. In *INDOCRYPT*, pages 10–17, 2007.

[144] D. Watanabe. OFFICIAL COMMENT: LUX. NIST mailing list, 2009. Available at `http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/documents/LUX_Comments.pdf`.

[145] S. Indesteege, F. Mendel, B. Preneel, and C. Rechberger. Collisions and Other Non-random Properties for Step-Reduced SHA-256. In Avanzi et al. [159], pages 276–293.

[146] F. Mendel, N. Pramstaller, C. Rechberger, and V. Rijmen. Analysis of Step-Reduced SHA-256. In *FSE*, pages 126–143, 2006.

[147] I. Nikolic and A. Biryukov. Collisions for Step-Reduced SHA-256. In *FSE*, pages 1–15, 2008.

[148] A. Biryukov, M. Lamberger, F. Mendel, and I. Nikolic. Second-Order Differential Collisions for Reduced SHA-256. In *ASIACRYPT*, pages 270–287, 2011.

[149] E. Biham and A. Shamir. Differential Cryptanalysis of the Full 16-Round DES. In *CRYPTO*, pages 487–496, 1992.

[150] T. Shirai and K. Araki. On Generalized Feistel Structures Using the Diffusion Switching Mechanism. *IEICE Transactions*, 91-A(8):2120–2129, 2008.

[151] T. Suzaki and K. Minematsu. Improving the Generalized Feistel. In *FSE*, pages 19–39, 2010.

[152] J. Kim, S. Hong, J. Sung, C. Lee, and S. Lee. Impossible Differential Cryptanalysis for Block Cipher Structures. In *INDOCRYPT*, pages 82–96, 2003.

[153] A. Biryukov and I. Nikolic. Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In *EUROCRYPT*, pages 322–344, 2010.

[154] E. Biham, O. Dunkelman, and N. Keller. Related-Key Impossible Differential Attacks on 8-Round AES-192. In *CT-RSA*, pages 21–33, 2006.

[155] B. Preneel, editor. *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*, volume 1008 of *Lecture Notes in Computer Science*. Springer, 1995.

[156] L. R. Knudsen, editor. *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*. Springer, 1999.

[157] A. Biryukov, G. Gong, and D. R. Stinson, editors. *Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers*, volume 6544 of *Lecture Notes in Computer Science*. Springer, 2011.

[158] M. Matsui, editor. *Fast Software Encryption, 8th International Workshop, FSE 2001 Yokohama, Japan, April 2-4, 2001, Revised Papers*, volume 2355 of *Lecture Notes in Computer Science*. Springer, 2002.

[159] R.M. Avanzi, L. Keliher, and F. Sica, editors. *Selected Areas in Cryptography, 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, volume 5381 of *Lecture Notes in Computer Science*. Springer, 2009.

[160] A. Biryukov, editor. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007.

List of Publications

Journal

1. Takanori Isobe, Toshihiro Ohigashi, Hidenori Kuwakado, and Masakatu Morii, "A Chosen-IV Key Recovery Attack on Py and Pypy", IEICE Trans. on Information and Systems, vol. E92-D, no.1, pp.32-40, 2009.

2. Takanori Isobe, "A Single Key Attack on the Full GOST Block Cipher", Journal of Cryptology, vol. 26(1), pp. 172-189, 2013.

3. Takanori Isobe, Toshihiro Ohigashi, and Masakatu Morii, "Slide Property of RAKAPOSHI and Its Application to Key Recovery Attack", Journal of Information Processing Information Processing Society of Japan, 2013, to apper.

4. Takanori Isobe, Toshihiro Ohigashi Yuhei Watanabe, and Masakatu Morii, "Comprehensive Analysis of Initial Keystream Biases of RC4", IEICE Trans. on Fundamentals of Electronics, Comm. and Computer Sciences, 2013, to appear.

International Conference

1. Takanori Isobe, Toshihiro Ohigashi, Hidenori Kuwakado, and Masakatu Morii, "How to Break Py and Pypy by a Chosen-IV Attack", Workshop Record of The State of the Art of Stream Ciphers (SASC) 2007, pp. 340-352, 2007.

2. Takanori Isobe and Kyoji Shibutani, "Preimage Attack on Reduced Tiger and SHA-2", Fast Software Encryption (FSE) 2009, Lecture Note in Computer Science, vol. 5665, pp. 139-155, Springer, 2009.

3. Takanori Isobe, "A Single Key Attack on the Full GOST Block Cipher", Fast Software Encryption (FSE) 2011, Lecture Note in Computer Science, vol. 6733, pp. 290-305, Springer, 2011.

4. Li Ji, Takanori Isobe, and Kyoji Shibutani, "Converting Meet-in-the-Middle Preimage Attack into Pseudo Collision Attack : Application to SHA-2", Fast Software Encryption (FSE) 2012, Lecture Note in Computer Science, vol. 7549, pp. 264-287. Springer, 2011.

5. Takanori Isobe and Kyoji Shibutani, "Security Analysis Lightweight Block Ciphers XTEA, LED and Piccolo", Australasian Conference on Information

Security and Privacy (ACISP) 2012, Lecture Note in Computer Science, vol. 7372, pp. 71-86, Springer, 2012.

6. Takanori Isobe and Kyoji Shibutani, "All Subkey Recovery Attack on Block Cipher:Extending Meet-in-the-Middle Approach", Selected Areas in Cryptography, the Conference on Selected Areas in Cryptography (SAC) 2012, Lecture Note in Computer Science, vol. 7707, pp. 202-221, Springer, 2012.

7. Takanori Isobe, Toshihiro Ohigashi, and Masakatu Morii, "Slide Cryptanalysis on Lightweight Stream Cipher RAKAPOSHI", International Workshop on Security, (IWSEC)2012, Lecture Note in Computer Science, vol. 7631, pp. 138-155, Springer, 2012.

8. Takanori Isobe, Yu Sasaki, and Jiageng Chen, "Related-Key Boomerang Attacks on KATAN32/48/64", Australasian Conference on Information Security and Privacy (ACISP) 2013, Lecture Note in Computer Science, vol. 7959, pp. 268-285, Springer, 2013.

9. Takanori Isobe, Toshihiro Ohigashi Yuhei Watanabe, and Masakatu Morii, "Full Plaintext Recovery Attack on Broadcast RC4", Fast Software Encryption, (FSE) 2013, Lecture Note in Computer Science, Springer, 2013, to appear.

10. Takanori Isobe, "Recent Meet-in-the-Middle Attack on Block Ciphers", The Second Asian Workshop on Symmetric Key Cryptology (ASK) 2012, Invited talk, `http://web.spms.ntu.edu.sg/~ask/2012/slides_03_Takanori_Isobe.pdf`, 2012.