Comprehensive study of integral analysis on LBlock

Sasaki, Yu; Wang, Lei

2014

Sasaki, Y., & Wang, L. (2014). Comprehensive Study of Integral Analysis on LBlock. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E97.A(1), 127-138.

<https://hdl.handle.net/10356/103921>

<https://doi.org/10.1587/transfun.E97.A.127>

© 2014 The Institute of Electronics, Information and Communication Engineers. This paper was published in IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences and is made available as an electronic reprint (preprint) with permission of The Institute of Electronics, Information and Communication Engineers. The paper can be found at the following official DOI: [<http://dx.doi.org/10.1587/transfun.E97.A.127>]. One print or electronic copy may be made for personal use only. Systematic or multiple reproduction, distribution to multiple locations via electronic or other means, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper is prohibited and is subject to penalties under law.

Comprehensive Study of Integral Analysis on LBlock*

Yu SASAKI^{†a)}, Member and Lei WANG^{††b)}, Nonmember

SUMMARY The current paper presents an integral cryptanalysis in the single-key setting against light-weight block-cipher LBlock reduced to 22 rounds. Our attack uses the same 15-round integral distinguisher as the previous attacks, but many techniques are taken into consideration in order to achieve comprehensive understanding of the attack; choosing the best balanced-byte position, meet-in-the-middle technique to identify right key candidates, partial-sum technique, relations among subkeys, and combination of the exhaustive search with the integral analysis. Our results indicate that the integral cryptanalysis is particularly useful for LBlock like structures. At the end of this paper, which factor makes the LBlock structure weak against the integral cryptanalysis is discussed. Because designing light-weight cryptographic primitives is an actively discussed topic, we believe that this paper returns some useful feedback to future designs.

key words: LBlock, integral analysis, partial-sum, meet-in-the-middle, subkey relations

1. Introduction

Block-ciphers are basic tools for secure communications which provide the confidentiality of the data. Recently, block-ciphers which can be implemented in resource constraint environment, e.g., RFID Tags for a sensor network, have received much attention. Such block-ciphers are called light-weight block-ciphers.

Many light-weight block-ciphers were designed so far. Some examples are HIGHT [1] proposed at CHES 2006 which were standardized by ISO as a 64-bit block-cipher [2], and PRESENT [3] proposed at CHES 2007 and CLEFIA [4] proposed at FSE 2007, which were standardized by ISO for the lightweight cryptography [5]. Many other designs were proposed independently of the ISO standards e.g., LBlock [6] proposed at ACNS 2011, Piccolo [7] proposed at CHES 2011, LED [8] proposed at CHES 2011, and TWINE [9] proposed at SAC 2012. Different designs provide different implementation characteristics, e.g., different tradeoff of area, throughput, and security, thus making a comparison and identifying good designs is very hard. Particularly security evaluation is hard because it takes long and

usually requires evaluations by the third party.

Integral analysis is a cryptanalytic technique against symmetric-key primitives, which was firstly proposed by Daemen et al. to evaluate the security of SQUARE cipher [10], and was later unified as *integral analysis* by Knudsen and Wagner [11]. The crucial part is a construction of an *integral distinguisher*: an attacker prepares a set of plaintexts which contains all possible values for some bytes and has a constant value for the other bytes. All plaintexts in the set are passed to an encryption oracle. Then, the corresponding state after a few rounds has a certain property, e.g. the XOR of all texts in the set becomes 0 with probability 1. Throughout the paper, this property is called *balanced*. A key recovery attack can be constructed by using this property. The attacker appends a few rounds to the end of the distinguisher. After she obtains a set of the ciphertexts, she guesses a part of round keys and performs the partial decryption up to the balanced state. If the guess is correct, the XOR sum of the results always becomes 0. Hence, the key space can be reduced.

Several improved techniques are known for the integral analysis. Ferguson et al. proposed a technique called *partial-sum* [12]. It reduces the complexity of the partial decryption up to the balanced state by guessing each subkey byte one after another. Sasaki and Wang introduced *meet-in-the-middle techniques* for the key recovery phase of the integral analysis against block-ciphers with a Feistel network [13]. It separates the partial decryption into two independent parts, and thus the complexity can be reduced.

The integral analysis has already been applied to LBlock. Firstly, the designers proposed 15-round integral distinguisher, and constructed an 18-round attack [6]. Then, Sasaki and Wang extended the attack up to 20 rounds [13]. Regarding other approaches, the designers proposed a 20-round impossible differential attack [6]. This was later extended up to 21 rounds by Liu et al. [14] and by Karakoç et al. [15]. These are the previous best results on LBlock in the single-key setting. For related-key attacks, Minier and Naya-Plasencia proposed a related-key impossible differential attack up to 22 rounds [16]. A paper for related-key differential attack by Liu et al. appeared in the accepted papers list of ICICS 2012 [17]. An optimization of the brute-force attack by the biclique technique was studied by Wang et al. at WISA 2012 [18]. In this paper, we do not discuss such an optimized brute force attack with a small advantage of the constant factor. Note that Karakoç et al. [15] also show a 22-round impossible differential attack in the single-key set-

Manuscript received March 29, 2013.

Manuscript revised July 12, 2013.

[†]The author is with NTT Secure Platform Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.

^{††}The author is with the Nanyang Technological University, Singapore.

*The preliminary version was presented in ICISC 2012. More detailed procedure for the 22-round attack is provided in this article. All the details for the 21-round attack are newly described for the comprehensive understanding.

a) E-mail: sasaki.yu@lab.ntt.co.jp

b) E-mail: Wang.Lei@ntu.edu.sg

DOI: 10.1587/transfun.E97.A.127

Table 1 Comparison of attack results.

Model	Approach	#Rounds	Data	Time	Memory (bytes)	Reference
Single-key	Imp. Diff.	20	2^{63}	$2^{72.7}$	2^{68}	[6]
	Imp. Diff.	21	$2^{62.5}$	$2^{73.7}$	$2^{55.5}$	[14]
	Imp. Diff.	21	2^{63}	$2^{69.5}$	2^{68}	[15]
	Integral	18	2^{62}	2^{36}	2^{20}	[6]
	Integral	20	$2^{63.6}$	$2^{39.6}$	2^{35}	[13]
	Integral	22	$2^{61.6}$	$2^{71.2}$	not given	[19] †
	Integral	21	$2^{61.6}$	$2^{54.16\ddagger}$	$2^{51.58}$	This paper
	Integral	22	2^{61}	$2^{70.00}$	2^{63}	This paper
Related-key	Zero correlation	22	$2^{62.1}$	$2^{71.27}$	2^{64}	[21]
	Imp. Diff.	22	2^{40}	2^{70}	not given	[16]

†: Unpublished independent work available only in Chinese.

‡: The attack requires $2^{61.58}$ memory access in order to process $2^{61.58}$ ciphertexts.

ting with a time complexity of $2^{79.28}$ LBlock computations. Due to the too expensive complexity, we do not compare the results in this paper.

Wang et al. [18], in their paper, cited a paper claiming a 22-round integral attack on LBlock [19]. As is written in the reference of [18], the paper seems unpublished and moreover available only in Chinese language. We emphasize that our attack is independent of [19], and our attack complexity is slightly better than the one in [19]. Although we have no idea about the details of [19], we believe that our comprehensive analysis gives several feedbacks to the cryptographic community, e.g., how to apply the integral analysis on block ciphers, how to analyze the key schedule function of LBlock, and the difficulty of extending the integral analysis on LBlock to 23 rounds.

1.1 Our Contributions

In this paper, we present a comprehensive study of the integral analysis against LBlock. Our goal is extending the number of attacked rounds and optimize the complexity as much as possible by considering all previously known techniques. Specifically, we consider the following techniques;

- There are 4 possibilities of the balanced-byte position at the output of the integral distinguisher. We try all of them to identify the best choice.
- We optimize the complexity by using the meet-in-the-middle approach.
- We optimize the complexity by using the partial-sum technique.
- We analyze the key schedule function, and exploit subkey relations.
- We combine the exhaustive search with integral analysis. This can optimize the data complexity.

As a result, we construct a 21-round attack with (Data, Time, Memory) = $(2^{61.6}, 2^{54.16}, 2^{51.58})$, which is better than the previous 21-round impossible differential attack with (Data, Time, Memory) = $(2^{62.5}, 2^{73.7}, 2^{55.5})$. We then extend the attack by one more round, and obtain a 22-round attack with (Data, Time, Memory) = $(2^{61}, 2^{70.00}, 2^{63})$. The attack results are summarized in Table 1.

The 15-round integral distinguisher discovered by the designers [6] produces the balanced byte at 4 byte-positions, 0th, 2nd, 4th, and 6th bytes of the intermediate state after 15 rounds. The previous integral attacks [6], [13] used the balanced byte at the 4th byte without any reasoning. Our analysis shows that the choice of the balanced-byte position is very sensitive when subkey relations are considered. Interestingly, as later explained in Table 2, using the balanced byte at the 6th byte for attacking 21 rounds and at the 2nd byte for attacking 22 rounds achieves significantly smaller complexity than the other 3 choices.

Our results indicate that the integral cryptanalysis is particularly useful for LBlock like structures. Indeed, LBlock is the almost only case that the integral cryptanalysis works more rounds than the impossible differential cryptanalysis. More discussion about the comparison of the integral cryptanalysis and the impossible differential cryptanalysis against the LBlock-like structure can be seen in [20]. Because designing light-weight cryptographic primitives is an actively discussed topic, we hope that this paper returns some useful feedback to future designs.

1.2 Follow-Up Work

After the submission of the preliminary version of this paper, an attack on 22-round LBlock in the single-key setting was reported by Soleimany and Nyberg [21]. Their approach is the zero-correlation linear cryptanalysis, and the attack complexity is (Data, Time, Memory) = $(2^{62.1}, 2^{71.27}, 2^{64})$. As you can see, our attack is better than [21].

1.3 Paper Outline

This paper is organized as follows. Section 2 gives preliminaries. Section 3 explains an observation to introduce some tradeoff between data and time complexities. Section 4 explains our 22-round attack on LBlock. In Sect. 5, we discuss which structure of LBlock affects the impact of integral analysis, and conclude the paper. Appendix A gives details of the 22-round attack. Appendix B shows the data for our 21-round attack.

2. Preliminaries

2.1 LBlock Specification

LBlock is a light-weight block-cipher proposed by Wu and Zhang [6]. The block size is 64 bits and the key size is 80 bits. It adopts a modified Feistel structure with 32 rounds, and its round function consists of the subkey addition, S-box transformations, and a permutation of the byte positions (1 byte is 4 bits).

Let $X_i^L || X_i^R$, where $0 \leq i \leq 32$, be an internal state which is an input to the i -th round or an output from the $i - 1$ -th round. We further denote 8 bytes inside of X_i^L and X_i^R by $X_i^L = X_i^L[7] || X_i^L[6] || \dots || X_i^L[0]$ and $X_i^R = X_i^R[7] || X_i^R[6] || \dots || X_i^R[0]$, respectively. The plaintext is loaded into an internal state $X_0^L || X_0^R$. The state $X_i^L || X_i^R$ is updated with a 32-bit subkey $K_i = K_i[7] || K_i[6] || \dots || K_i[0]$ by the following equations.

$$X_{i+1}^L = P(S(X_i^L \oplus K_i)) \oplus (X_i^R \lll 8), \quad X_{i+1}^R = X_i^L,$$

where, $S(\cdot)$, $P(\cdot)$, and $\lll 8$ represent an S-box layer, a permutation of the byte positions, and the left cyclic shift by 8 bits, respectively. In the S-box layer, each byte is updated according to the 4-bit to 4-bit S-boxes defined in the specification. Then, $P(x_7 || x_6 || x_5 || x_4 || x_3 || x_2 || x_1 || x_0)$ returns $(x_6 || x_4 || x_7 || x_5 || x_2 || x_0 || x_3 || x_1)$. These computations are described in Fig. 1. In this paper, we denote the state after the byte-position permutation in round i by Z_i . After 32 rounds, $X_{32}^L || X_{32}^R$ are produced as the ciphertext.

2.1.1 Key Schedule Function

The key schedule function produces thirty-two 32-bit subkeys from an 80-bit secret key. Let κ_i , where $0 \leq i \leq 31$, be an 80-bit internal state for the key schedule function for round i . We denote each bit of κ_i by $\kappa_i[79], \kappa_i[78], \dots, \kappa_i[0]$. We often denote several bits of κ_i by $\kappa_i[a, b, c, \dots]$.

The leftmost 32 bits of κ_0 , i.e., $\kappa_0[79, 78, \dots, 48]$, are output as a 32-bit subkey for round 0, K_0 . Then, the following is operated for $i = 1, 2, \dots, 31$.

1. $\kappa_i \leftarrow \kappa_{i-1} \lll 29$.

Fig. 1 LBlock round function.

2. Update $\kappa_i[79, 78, 77, 76]$ and $\kappa_i[75, 74, 73, 72]$ by $S_9(\kappa_i[79, 78, 77, 76])$ and $S_8(\kappa_i[75, 74, 73, 72])$ respectively, where S_8 and S_9 are 4-bit to 4-bit S-boxes.
3. Update $\kappa_i[50, 49, 48, 47, 46]$ by $\kappa_i[50, 49, 48, 47, 46] \oplus [i]_2$, where $[i]_2$ is a binary representation of the round index.
4. Output the leftmost 32 bits of κ_i as a 32-bit subkey K_i .

2.2 Notations for Integral Attack

The integral attack is a cryptanalytic technique for symmetric-key primitives. Its brief description has already given in Sect. 1. To discuss integral distinguishers, the following notations are used in this paper.

“A (Active)”: all values appear exactly the same number in the set of texts.

“B (Balanced)”: the XOR of all texts in the set is 0.

“C (Constant)”: the value is fixed to a constant for all texts in the set.

We also use the following notations to describe the attack.

D : number of plaintexts to construct an integral distinguisher.

K_α : size of subkeys (in bits) recovered during the key recovery phase.

K_β : size of subkeys (in bits) exhaustively guessed after the key recovery phase.

$|B|$: size of the balanced state (in bits) to be checked in the key recovery phase. The key space is reduced by $|B|$ bits with analyzing a single set.

The previous integral attack, especially for LBlock, analyzed $K_\alpha/|B|$ plaintext sets to identify the right key of K_α . Then, K_β is recovered by the exhaustive search. The data complexity is $D \cdot (K_\alpha/|B|)$ and the time complexity is a sum of the one for the key recovery phase and 2^{K_β} . Several techniques can be applied to reduce the time complexity of the key recovery phase. Note that if D is much bigger than the time complexity for the key recovery phase and the exhaustive search, $D \cdot (K_\alpha/|B|)$ memory access for processing obtained ciphertexts is the bottle-neck of the time complexity.

2.3 Partial-Sum Technique

The partial-sum technique was introduced by Ferguson et al. [12] in order to improve the complexity of the key recovery phase in the integral attack. The original attack target was AES. In the key recovery phase of the AES, the partial decryption involves 5 bytes of the key and 4 bytes of the ciphertext. Suppose that the number of data to be analyzed, n , is 2^{32} and the byte position b of each ciphertext is denoted by $C_{b,n}$. Then, the equation can be described as follows.

$$\bigoplus_{n=1}^{2^{32}} \left[S_4(S_0(c_{0,n} \oplus k_0) \oplus S_1(c_{1,n} \oplus k_1) \oplus \dots \oplus S_7(c_{7,n} \oplus k_7)) \right]$$

Fig. 2 15-round integral distinguisher on LBlock.

$$S_2(c_{2,n} \oplus k_2) \oplus S_3(c_{3,n} \oplus k_3) \oplus k_4 \Big]. \quad (1)$$

With a straightforward method, the analysis takes $2^{32+40} = 2^{72}$ partial decryptions, while the partial-sum technique can perform this computation only with 2^{48} partial decryptions. The idea is partially computing the sum by guessing each key byte one after another.

The analysis starts from 2^{32} texts $(c_{0,n}, c_{1,n}, c_{2,n}, c_{3,n})$. First, two key bytes k_0 and k_1 are guessed, and $S_0(c_{0,n} \oplus k_0) \oplus S_1(c_{1,n} \oplus k_1)$ is computed for each guess. Let $x_{i,n}$ be $\bigoplus_{p=0}^i (S_p(c_{p,n} \oplus k_p))^\dagger$. Then, $S_0(c_{0,n} \oplus k_0) \oplus S_1(c_{1,n} \oplus k_1)$ can be represented by $x_{1,n}$, and Eq. (1) becomes

$$\bigoplus_{n=1}^{2^{32}} \left[S_4 \left(x_{1,n} \oplus S_2(c_{2,n} \oplus k_2) \oplus S_3(c_{3,n} \oplus k_3) \oplus k_4 \right) \right].$$

The original set includes 2^{32} texts, but now only 3-byte information (x_1, c_2, c_3) is needed. Hence, by counting how many times each of 3-byte values (x_1, c_2, c_3) appears and by only picking the values that appear odd times, the size of the data set is compressed into 3 bytes. For the second step, a single key byte k_2 is guessed, and the size of the data set becomes 2 bytes (x_2, c_3) . For the third step, a single key byte k_3 is guessed, and the size of the data set becomes 1 byte (x_3) . Finally, a single byte k_4 is guessed and Eq. (1) is computed for each guess.

The complexity for the guess of k_0, k_1 is $2^{16} \times 2^{32} = 2^{48}$, for the guess of k_2 is $2^{16} \times 2^8 \times 2^{24} = 2^{48}$. Similarly, the complexity is preserved to be 2^{48} until the last computation.

2.4 Previous Integral Analysis on LBlock

The designers showed a 15-round integral distinguisher [6], which is shown in Fig. 2. For a set of 2^{60} plaintexts with the form of (AAAC AAAA AAAA AAAA), the state after 15 rounds, $(X_{15}^L || X_{15}^R)$, has the form of (???? ???? ?B?B ?B?B). By using this property, the designers showed an 18-round key recovery attack. The attacker guesses a part of subkeys, and performs the partial decryption up to the fourth byte of X_{15}^R and checks if its sum is 0 or not. The partial decryption up to $X_{15}^R[4]$ involves 5 bytes of the ciphertext and 4 bytes of subkeys. The attacker first counts how many times each 5-byte value of the ciphertext appears and only picks the ones that appear odd times. Hence, at most $2^{4 \times 5} = 2^{20}$ values are stored in a memory. Then, for each guess of four key bytes, she computes the corresponding $X_{15}^R[4]$ and computes

Fig. 3 20-round attack on LBlock.

the sum. The attack complexity is $2^{20} \times 2^{16} = 2^{36}$ partial decryptions. With the analysis of a single 2^{60} plaintexts set, the key space is reduced by 1 byte. Therefore, to identify the right key, 4 sets of plaintexts need to be analyzed. Hence, the data complexity is $4 \times 2^{60} = 2^{62}$.

Sasaki and Wang introduced a meet-in-the-middle technique for the key recovery phase of the integral analysis, and extended the number of attacked rounds up to 20 rounds [13]. The 5-round key-recovery phase is shown in Fig. 3. They utilize the property that $\bigoplus X_{15}^R[4] = 0$ can be written as $\bigoplus Z_{15}[6] = \bigoplus X_{16}^L[6]$. Therefore, the sum of $Z_{15}[6]$ and $X_{16}^L[6]$ can be computed independently, and right-key candidates are identified by checking their matches with the meet-in-the-middle technique. The partial decryption for $\bigoplus Z_{15}[6]$ involves 8 bytes of the ciphertext and 8 bytes of subkeys, and thus requires $2^{32} \times 2^{32} = 2^{64}$ partial decryptions. The partial decryption for $\bigoplus X_{16}^L[6]$ involves 5 bytes of the ciphertext and 4 bytes of subkeys, and thus only requires $2^{20} \times 2^{16} = 2^{36}$ partial decryptions. Moreover, they applied the partial-sum technique, and the complexity for $\bigoplus Z_{15}[6]$ was reduced into 2^{36} partial decryptions. In this attack, 12 key bytes are guessed, and the key space is reduced by 1 byte with the analysis of a single 2^{60} plaintexts set. Therefore, to identify the right key, 12 sets of plaintexts need to be analyzed. Hence, the data complexity is

[†]Notation $x_{i,n}$ represents the sum of i S-box outputs.

$12 \times 2^{60} = 2^{63.6}$, which is very close to the full code book.

3. Trade-Off between Data and Time by Combining Exhaustive Search

In this section, we explain a simple technique to optimize the attack. This gives the trade-off between the data complexity and time complexity. For example, we can convert the previous 20-round attack by Sasaki and Wang [13] with (Data, Time)=($2^{63.7}$, 2^{36}) into the one with (Data, Time)=(2^{62} , 2^{64}), and thus can avoid the marginal improvement of the data complexity. Note that the complexity evaluation by Sasaki and Wang [13] is only for the key recovery phase. Their attack, besides 2^{36} computations, requires $2^{63.7}$ memory access to process $2^{63.7}$ ciphertexts.

The approach is very simple. When we recover a part of subkeys, we stop identifying the unique right key for K_α , but reduce the key space into a sufficiently small size. Then, the reduced key space is exhaustively searched together with the remaining subkey bits K_β . Note that the exhaustive search only for K_α (independently of K_β) is impossible. K_α and K_β must be guessed together. With this method, the data complexity can be reduced with an extra cost for the exhaustive search. Let d be the number of sets to be analyzed. Then, the data complexity is $d \cdot D$. The key space for K_α is reduced into $K_\alpha - (d \cdot |B|)$, and the cost for the exhaustive search becomes $2^{K_\alpha - (d \cdot |B|) + K_\beta}$.

Let us apply this method to the previous 20-round attack [13] with (Data, Time)=($2^{63.7}$, 2^{36}). More precisely, the parameters of this attack are $D = 2^{60}$, $K_\alpha = 48$, $K_\beta = 32$, $|B| = 4$. [13] chose $d = 12$, thus the data complexity is $12 \cdot 2^{60} = 2^{63.6}$ and the time complexity for the exhaustive search is $2^{0+32} = 2^{32}$. We now change the parameter d to $d = 4$. Then, the data complexity is $4 \cdot 2^{60} = 2^{62}$ and the time complexity for the exhaustive search is $2^{48-16+32} = 2^{64}$. Considering that the attack requires at least 2^{60} memory access to process the ciphertexts, the time complexity is now almost equally distributed, and we can avoid the marginal improvement of the data complexity.

In a later section, we use this method to optimize the attack.

4. 21-Round and 22-Round Attacks on LBlock

In this section, we extend the number of attacked rounds, and make as much effort as possible to optimize the attack. We did not find any improvement about the 15-round integral distinguisher. Hence, we use the same distinguisher as previous work which is shown in Fig. 2. Then, we consider the following techniques to optimize the attack.

- We try all possible balanced-byte positions. The previous attacks [6], [13], without any reasoning, used $X_{15}^R[4]$ to check if the partial decryption results in the balanced state. We try all of $X_{15}^R[0]$, $X_{15}^R[2]$, $X_{15}^R[4]$, and $X_{15}^R[6]$ to identify the best choice.
- Similarly to [13], we use the meet-in-the-middle approach to reduce the complexity.

- Similarly to [13], we use the partial-sum technique to reduce the complexity.
- Different from the previous attacks, we further analyze the key schedule function, and take into account the relation between subkeys. The analysis depends on which subkeys we guess. Hence, the analysis must be performed for all possible balanced-byte positions in X_{15}^R .
- Finally, we consider combining the exhaustive search for K_α discussed in Sect. 3.

The second and third items were already considered in [13], and we confirmed that the attack could not be extended only with these two techniques. Hence, the other three techniques which are newly considered in this paper essentially contribute to extend the number of attacked rounds.

4.1 Overview without Considering the Key Schedule Function

As was done in [13], to detect the right key candidates, the attacker computes the sum of the target byte in Z_{15} and the sum of the target byte of X_{16}^L independently, and find matches between two results. Due to the Feistel structure, the bottle-neck of the time complexity is the one for Z_{15} . Hence, how many subkey bytes relate to the computation for Z_{15} is important. The number of total subkey bytes, K_α , is also important to estimate the number of necessary text sets.

We firstly obtain such information for each of the balanced-byte position, $X_{15}^R[0]$, $X_{15}^R[2]$, $X_{15}^R[4]$, and $X_{15}^R[6]$. As a result of the analysis, we found that such important factors are the same for all balanced-byte positions. In details, for 21-round attack, the computation for Z_{15} involves 13 subkey bytes and 12 ciphertext bytes, and K_α is 80 (20 bytes). For 22-round attack, the computation for Z_{15} involves 20 subkey bytes and 15 ciphertext bytes, and K_α is 128 (32 bytes).

21- and 22-round attacks are impossible only with the techniques in the previous attacks. We then analyze the key schedule function and exploit the relation between subkeys. If it is considered, the attack complexity is very different depending on the balanced-byte position.

4.2 Analysis of Key Schedule Function

What we do here is guessing several bits of the key state, and trace the guessed bit positions during several rounds. If the guess of several subkey bits reveals some information about other subkeys in different rounds, the number of guessed bits by the attacker can be reduced.

First of all, we summarize the property of the LBlock key schedule function.

- Three kinds of operations are used in the key schedule function, i.e., cyclic shift, S-box transformations, and constant XOR. The cyclic shift and constant XOR do not change the number of known bits of the key state.

Table 2 Summary of key space for 21- and 22-round attacks for all balanced-byte positions.

Balanced-byte position	21-round attack		22-round attack	
	Key space for Z_{15}	K_α	Key space for Z_{15}	K_α
$X_{15}^R[0]$	50	63	62	75
$X_{15}^R[2]$	44	61	55	69
$X_{15}^R[4]$	47	63	63	75
$X_{15}^R[6]$	42	57	65	77

The number of known bits is reduced only if the S-box transformation takes several known bits and unknown bits together as input.

- The number of known bits is reduced quickly in the forward direction. Suppose that the attack needs to guess $K_i[0]$, which are 4 bits of $\kappa_i[51, 50, 49, 48]$. These go to $\kappa_{i+1}[79, 78, 77, 0]$ by the left 29-bit cyclic shift. Then, the S-box transformation is applied to $\kappa_{i+1}[79, 78, 77, 76]$. Because $\kappa_{i+1}[76]$ is unknown, the attacker cannot compute the output of the S-box, and lose 3-bit information.
- The number of known bits is reduced slowly in the backward direction. Suppose that the attack needs to guess $K_{i+1}[7]$, which are 4 bits of $\kappa_{i+1}[79, 78, 77, 76]$. By following the inverse of the key schedule, these go to the S-box transformation. Because the attacker knows all information, the inverse of the S-box can be computed and the attacker does not lose any information. Then, $\kappa_{i+1}[79, 78, 77, 76]$ moves to $\kappa_i[51, 50, 49, 48]$, and the attacker still knows 3 bits of $K_i[0]$.

Thanks to the slow decrease of the number of known bits in backward, the subkey guesses for the last several rounds can efficiently be used to know subkeys in earlier rounds.

We then analyze the subkey relations for both of 21- and 22-rounds and for each balanced-byte position. As shown in Sect. 4.1, the number of guessed key bytes is too many, and thus we should choose the balanced-byte position with minimum key space. The results of the analysis is summarized in Table 2. The columns for “Key space for Z_{15} ” show how many key bits must be guessed to compute the sum of the target byte in Z_{15} by considering the subkey relations, which is the bottle-neck of the time complexity. The columns for “ K_α ” show how many bits are guessed to analyze a single plaintext set, i.e., K_α is the number of elements of the union of the key space for Z_{15} and the key space for X_{16}^L . A smaller number indicates that overlaps of subkeys occur more frequently, and thus the number of guessed bits can be small. From Table 2, using $X_{15}^R[6]$ and $X_{15}^R[2]$ as the balanced-byte position would yield the best attack for 21 rounds and 22 rounds, respectively. It is particularly interesting that K_α for 22-round attack is significantly smaller for $X_{15}^R[2]$ than other balanced-byte positions.

In the next section, we explain the detailed attack procedure. Because procedures for 21 rounds and 22 rounds are very similar, we only describe the 22-round attack in details. The 21-round attack is explained in Appendix B. We

Fig. 4 7-round key-recovery phase.

also explain the detailed analysis of the key schedule function in the next section only for the 22-round attack.

4.3 7-Round Key-Recovery Phase for 22-Round Attack

Details of the 7-round key-recovery phase are shown in Fig. 4. We follow the notations used in [13], where 20 key bytes and 15 ciphertext bytes related to the computation of $Z_{15}[4]$ are denoted by numbers in black square brackets. Similarly, 12 key bytes and 12 ciphertext bytes related to the computation of $X_{16}^L[4]$ are denoted by numbers in grey round brackets. The sum of $Z_{15}[4]$ and $X_{16}^L[4]$ are computed independently, and the bottle-neck is the computation for $Z_{15}[4]$. Hereafter, we mainly explain how to compute the sum of $Z_{15}[4]$.

4.3.1 High-Level Description of the Attack

Because the attack procedure is very complicated, we first give the high-level description of the attack. The attack complexity depends on how we apply the partial-sum technique and how we utilize the subkey relations. In our attack, there are 2 patterns of the application of the partial-sum.

1-byte partial-sum: Suppose that we compute the partial-sum for a target byte of X_i^R . This computation always involves 1 byte of K_i and 2 bytes of (X_{i+1}^L, X_{i+1}^R) . The 1-byte partial-sum can be applied when the 2 bytes of (X_{i+1}^L, X_{i+1}^R) only relate to the computation of a target byte of X_i^R . Namely, after we obtain the sum of X_i^R , we can discard the information on the 2 bytes of (X_{i+1}^L, X_{i+1}^R) . An example of the 1-byte partial-sum is shown in Fig. 5, which we compute $X_{21}^R[0] = X_{20}^L[0] = (S_0(X_{22}^L[0] \oplus K_{21}[0]) \oplus X_{22}^L[2]) \ggg 8$, and $X_{22}^R[0]$ and $X_{22}^L[2]$ are only used to compute $X_{21}^R[0]$. Suppose, the number of texts to be analyzed is 2^{4N} , which consists of N -byte information ($X_{22}^R[0]$, $X_{22}^L[2]$, and other $N-2$ bytes). The attacker exhaustively guesses 4 bits of $K_{21}[0]$, and for each guess, computes $X_{21}^R[0]$ for all 2^{4N} texts. The attacker counts how many times each $N-1$ byte tuple ($X_{21}^R[0]$ and other $N-2$ bytes) appears, and only picks the ones which appear odd times. The analysis requires 2^{4N+4} 1-round computations, and for each guess, the data is compressed into 2^{4N-4} . Therefore, the complexity of 2^{4N+4} is preserved until the end.

3-byte partial-sum: Suppose that we need to compute X_i^R by using 2-byte information of (X_{i+1}^L, X_{i+1}^R) . The 3-byte partial-sum is applied when X_{i+1}^R is used not only for computing the target byte of X_i^R but also for computing one byte of X_{i-1}^R . In this case, the data cannot be compressed after we guess 4 bits of K_i , and thus the attack complexity increases. An example of the 3-byte partial-sum is shown in Fig. 6, which $X_{22}^R[1, 7]$ are used not only for computing $X_{21}^R[6, 3]$ but also for computing $X_{20}^R[7, 5]$. We first guess two bytes of $K_{21}[1, 7]$, and update the value of $X_{22}^L[0, 5]$ to $X_{21}^R[6, 3]$. This requires the complexity of 2^{4N+8} . Then, 1-byte partial-sum is applied for the computation of $X_{20}^R[5]$ (and then $X_{20}^R[7]$). In summary, the 3-byte partial-sum increases the complexity into $2^{(4N+8)+4}$.

The attack optimization also requires to consider the subkey relations, i.e., we need to arrange the computation order so that we can reduce the number of guessed bits by using the relations. This also makes the attack complicated. In our attack, these techniques are considered simultaneously by hand. In high-level, the complexity for computing the sum of $Z_{15}[4]$ can be explained as follows.

- The analysis starts from 2^{60} texts, 15-byte information of the ciphertext.
- The 1-byte partial-sum is applied several times. At this stage, the complexity is preserved to be $2^{60+4} = 2^{64}$

Fig. 5 Partial-sum with 1-byte guess.

Fig. 6 Partial-sum with 3-byte guess.

1-round computations.

- The 3-byte partial-sum is applied once, but thanks to the subkey relations, we can save the guess of 3 bits to update the value. At this stage, the complexity increases to $2^{(60+5)+4} = 2^{69}$ 1-round computations.
- Thanks to the subkey relations, the 1-byte partial-sum is applied by only guessing 2 bits. This occurs 2 times. At this stage, the complexity is $2^{(60+5-4)+4} = 2^{65}$ 1-round computations.
- The 3-byte partial-sum is applied once again. At this stage, the complexity increases to $2^{(60+5-4+8)+4} = 2^{73}$ 1-round computations.
- The remaining part can be computed with less complexity due to the subkey relations. Hence, intuitively, the bottle-neck of the complexity is 2^{73} 1-round computations.

4.3.2 Detailed Attack Procedure

The attack procedure to compute $\bigoplus Z_{15}[4]$ for a single text set is as follows. Its summary is available in Table A.1 in Appendix A. The key state for the last seven rounds, $K_{15}, K_{16}, \dots, K_{21}$ are described in Fig. 7.

1. Query 2^{60} plaintexts which has the form of (AAAC AAAA AAAA AAAA).
2. Prepare the memory which stores how many times each fifteen-byte value $X_{22}^L[0, 1, 2, 4, 5, 6, 7]$, $X_{22}^R[0, 1, 2, 3, 4, 5, 6, 7]$ appears, and pick the values which appear odd times.

Fig. 7 Key state for the last 7 rounds.

3. Guess 4 bits of $K_{21}[0]$, and compute $X_{21}^R[0]$ with $X_{22}^L[2], X_{22}^R[0]$. Compress the data into 2^{56} texts of $(X_{22}^L[0, 1, 4, 5, 6, 7], X_{22}^R[1, 2, 3, 4, 5, 6, 7], X_{21}^R[0])$.
4. Guess 4 bits of $K_{21}[4]$, and compute $X_{21}^R[4]$ with $X_{22}^L[6], X_{22}^R[4]$. Compress the data into 2^{52} texts of $(X_{22}^L[0, 1, 4, 5, 7], X_{22}^R[1, 2, 3, 5, 6, 7], X_{21}^R[0, 4])$.
5. Guess 4 bits of $K_{21}[6]$, and compute $X_{21}^R[5]$ with $X_{22}^L[7], X_{22}^R[6]$. Compress the data into 2^{48} texts of $(X_{22}^L[0, 1, 4, 5], X_{22}^R[1, 2, 3, 5, 7], X_{21}^R[0, 4, 5])$.
6. Guess 4 bits of $K_{20}[0]$, and compute $X_{20}^R[0]$ with $X_{22}^R[2], X_{21}^R[0]$. Compress the data into 2^{44} texts of $(X_{22}^L[0, 1, 4, 5], X_{22}^R[1, 3, 5, 7], X_{21}^R[4, 5], X_{20}^R[0])$. The known (guessed) bits of the key state upto this step are labeled as 1 in Fig. 7.
7. Guess $K_{21}[7]$. From Fig. 7, this can be done by 1-bit guess of $\kappa_{20}[47]$, which is labeled as 2. Compute $X_{21}^R[3]$ with $X_{22}^L[5], X_{22}^R[7]$. Update the data into 2^{44} texts of $(X_{22}^L[0, 1, 4], X_{22}^R[1, 3, 5, 7], X_{21}^R[3, 4, 5], X_{20}^R[0])$.
8. Guess 4 bits of $K_{21}[1]$, and compute $X_{21}^R[6]$ with $X_{22}^L[0], X_{22}^R[1]$. Update the data into 2^{44} texts of $(X_{22}^L[1, 4], X_{22}^R[1, 3, 5, 7], X_{21}^R[3, 4, 5, 6], X_{20}^R[0])$.
9. Guess 4 bits of $K_{20}[3]$, and compute $X_{20}^R[7]$ with $X_{22}^R[1], X_{21}^R[3]$. Compress the data into 2^{40} texts of $(X_{22}^L[1, 4], X_{22}^R[3, 5, 7], X_{21}^R[4, 5, 6], X_{20}^R[0, 7])$. The new known bits of the key state upto this step are labeled as 2 in Fig. 7.
10. Guess $K_{19}[7]$. From Fig. 7, this can be done by 2-bit guess of $\kappa_{19}[78, 79]$, which is labeled as 3. Compute $X_{19}^R[3]$ with $X_{21}^R[5], X_{20}^R[7]$. Compress the data into 2^{36} texts of $(X_{22}^L[1, 4], X_{22}^R[3, 5, 7], X_{21}^R[4, 6], X_{20}^R[0], X_{19}^R[3])$.
11. Guess 4 bits of $K_{20}[6]$, and compute $X_{20}^R[5]$ with $X_{22}^R[7], X_{21}^R[6]$. Compress the data into 2^{32} texts of $(X_{22}^L[1, 4], X_{22}^R[3, 5], X_{21}^R[4], X_{20}^R[0, 5], X_{19}^R[3])$.
12. Guess $K_{19}[5]$. From Fig. 7, this can be done by 2-bit guess of $\kappa_{19}[68, 69]$, which is also labeled as 3. Compute $X_{19}^R[2]$ with $X_{21}^R[4], X_{20}^R[5]$. Compress the data into 2^{28} texts of $(X_{22}^L[1, 4], X_{22}^R[3, 5], X_{20}^R[0], X_{19}^R[2, 3])$.
13. Guess 4 bits of $K_{21}[3]$, and compute $X_{21}^R[7]$ with $X_{22}^L[1], X_{22}^R[3]$. Update the data into 2^{28} texts of $(X_{22}^L[4], X_{22}^R[3, 5], X_{21}^R[7], X_{20}^R[0], X_{19}^R[2, 3])$.
14. Guess 4 bits of $K_{21}[5]$, and compute $X_{21}^R[2]$ with $X_{22}^L[4], X_{22}^R[5]$. Update the data into 2^{28} texts of $(X_{22}^L[3, 5], X_{21}^R[2, 7], X_{20}^R[0], X_{19}^R[2, 3])$.
15. Guess 4 bits of $K_{20}[7]$, and compute $X_{20}^R[3]$ with $X_{22}^L[5], X_{21}^R[7]$. Compress the data into 2^{24} texts of $(X_{22}^L[3], X_{21}^R[2], X_{20}^R[0, 3], X_{19}^R[2, 3])$. The new known bits of the key state upto this step are also labeled as 3 in Fig. 7.
16. From Fig. 7, $K_{18}[2]$ is already known. Compute $X_{18}^R[1]$

Fig. 8 κ_{18} for computing $Z_{15}[4]$ and $X_{16}^L[4]$. Known bits are in grey.

with $X_{20}^R[3], X_{19}^R[2]$. Compress the data into 2^{20} texts of $(X_{22}^R[3], X_{21}^R[2], X_{20}^R[0], X_{19}^R[3], X_{18}^R[1])$.

17. Guess 4 bits of $K_{20}[2]$, and compute $X_{20}^R[1]$ with $X_{22}^R[3], X_{21}^R[2]$. Update the data into 2^{20} texts of $(X_{21}^R[2], X_{20}^R[0, 1], X_{19}^R[3], X_{18}^R[1])$.
18. From Fig. 7, $K_{18}[3]$ is already known. Compute $X_{18}^R[7]$ with $X_{20}^R[1], X_{19}^R[3]$. Compress the data into 2^{16} texts of $(X_{21}^R[2], X_{20}^R[0], X_{18}^R[1, 7])$.
19. Guess $K_{19}[0]$. From Fig. 7, this can be done by 1-bit guess of $\kappa_{19}[51]$, which is labeled as 4. Compute $X_{19}^R[0]$ with $X_{21}^R[2], X_{20}^R[0]$. Compress the data into 2^{12} texts of $(X_{19}^R[0], X_{18}^R[1, 7])$.
20. From Fig. 7, $K_{17}[1]$ is already known. Compute $X_{17}^R[6]$ with $X_{19}^R[0], X_{18}^R[1]$. Compress the data into 2^8 texts of $(X_{18}^R[7], X_{17}^R[6])$.
21. Guess $K_{16}[6]$. From Fig. 7, this can be done by 2-bit guess of $\kappa_{16}[73, 74]$, which is labeled as 4. Compute $X_{16}^R[5]$ with $X_{18}^R[7], X_{17}^R[6]$. Compress the data into 2^4 texts of $X_{16}^R[5]$.
22. Guess 4 bits of $K_{15}[5]$, and compute the final sum for $Z_{15}[4]$. We store the result together with the guessed key value in a list $L_{Z_{15}}$. The new known bits of the key state upto this step are also labeled as 4 in Fig. 7.

4.3.3 Complexity Estimation for $\bigoplus Z_{15}[4]$

Step 2 requires 2^{60} memory access to process 2^{60} ciphertexts. Note that we need a memory to store $2^{60} \cdot 60 \approx 2^{66}$ -bit information. Step 3 requires $2^{60} \cdot 2^4 = 2^{64}$ 1-round computations. Step 4 requires $2^{56} \cdot 2^8 = 2^{64}$ 1-round computations. Step 5 requires $2^{52} \cdot 2^{12} = 2^{64}$ 1-round computations. Similarly, the complexity is estimated as a product of the previous data size and the total number of guessed bits. The complexity for each step is shown in the last column of Table A.1. In the end, the complexity to compute $\bigoplus Z_{15}[4]$ is

$$\begin{aligned}
 & 2^{64} + 2^{64} + 2^{64} + 2^{64} + 2^{61} + 2^{65} + 2^{69} + 2^{67} + 2^{67} \\
 & + 2^{65} + 2^{65} + 2^{69} + 2^{73} + 2^{69} + 2^{69} + 2^{69} + 2^{66} + 2^{62} \\
 & + 2^{60} + 2^{60} \\
 & \approx 2^{73.46} \text{ 1-round computations.}
 \end{aligned}$$

After Step 22, we obtain a list $L_{Z_{15}}$ with 2^{56} entries which contains 60-bit information; $\bigoplus Z_{15}[4]$ and 56-bit key values.

4.3.4 Remaining Part of the Attack

The complexity for computing $\bigoplus X_{16}^L[4]$ must be estimated. It involves 12 key bytes and 12 ciphertext bytes, and thus the

partial-sum technique must be considered. We confirmed that $\bigoplus X_{16}^L[4]$ is computed with at most $12 \cdot 2^{48+8} < 2^{60}$ 1-round computations. Due to the limited space, we omit the details. We store the result of $\bigoplus X_{16}^L[4]$ together with the 48-bit guessed keys in a table $L_{X_{16}^L}$. After the analysis, we obtain a list $L_{X_{16}^L}$ with 2^{48} entries.

After we make two lists $L_{Z_{15}}$ and $L_{X_{16}^L}$, we identify the right key candidates by checking the match. By the condition $\bigoplus Z_{15}[4] = \bigoplus X_{16}^L[4]$, the key space can be reduced by 4 bits. Moreover, because the computations of $Z_{15}[4]$ and $X_{16}^L[4]$ share some key bits in common, we can reduce the key space more. The key state κ_{18} for computing $Z_{15}[4]$ and $X_{16}^L[4]$ are shown in Fig. 8. 25 key bits are overlapped between κ_{18} for computing $Z_{15}[4]$ and $X_{16}^L[4]$. Therefore, the key space can be reduced by $4 + 25 = 29$ bits in total. Note that 12 bits of κ_{18} are not included in both parts. Those 12 bits are exhaustively searched after the key space for other 68 bits of κ_{18} is sufficiently reduced.

Finally, we conclude the attack. We analyze 2 sets of 2^{60} texts. The key space for the guessed 68 bits of κ_{18} is reduced into $2^{56+48-(29 \cdot 2)} = 2^{46}$. These 2^{46} and the other 12 bits are exhaustively searched with $2^{46+12} = 2^{58}$ LBlock computations. The data complexity is 2^{61} plaintexts. The time complexity is $2 \cdot (2^{73.46} + 2^{60})$ round functions and 2^{58} 22-round LBlock computations. The cost for computing one round function is regarded as 1/22 of 22-round LBlock computations. Hence, the total cost is $2 \cdot (2^{73.46} + 2^{60})/22 + 2^{58} \approx 2^{70.00}$ 22-round LBlock computations. The memory complexity is 2^{66} bits, which is 2^{63} bytes.

Note that, in these days, cryptographers evaluate the complexity by only counting the ratio of the number of involved S-box transformations during the attack to the number of total S-box transformations in the entire algorithm. For example, [18] takes this approach for LBlock. In this paper, we did not evaluate the complexity of our attack in such a way because it is unclear how the complicated attack procedure affects the exact complexity. However, we emphasize that if the complexity is evaluated based on the ratio of the number of S-box transformations, our attack becomes much faster than the current evaluations[†].

[†]For comparison, we show the complexity based on the ratio of the S-box transformation. The dominant part of the complexity becomes $2^{73.46}$ S-box transformations instead of $2^{73.46}$ round functions. Because the 22-round encryption computes $8 * 22$ S-box transformation, $2^{73.46}$ S-box transformations is equivalent to $2^{73.46}/(8 * 22) \approx 2^{66.00}$ 22-round LBlock computations.

5. Discussion and Concluding Remarks

5.1 Discussion

In this section, we discuss which structure of LBlock affects the efficiency of the integral analysis. The partial decryption for the last seven rounds requires to analyze 2^{60} texts for 55 subkey bits. If the round function is ideal, the analysis would require $2^{60+55} = 2^{115}$ computations, which is more expensive than the exhaustive search. However, the partial-sum technique on LBlock allows us to guess only a few subkey bits and then reduce the data to be analyzed by the same factor, thus, the partial decryption can go many rounds. Such a situation is caused by the fact that each subkey byte affects to the internal state value very slowly, i.e., only 1 byte of the internal state after 1 round, 2 bytes after 2 rounds, 3 bytes after 3 rounds, and so on. In other words, if the impact of each subkey byte expands more quickly, the attack becomes inefficient.

Note that we are not claiming that the LBlock-like structure is weak. The entire structure can be strong by iterating the round function many times. Indeed, our attack cannot reach full rounds. Our important note here is that designers may prefer to take a larger security margin than ciphers with other structures take.

5.2 Concluding Remarks

In this paper, we presented a comprehensive study of the integral analysis against LBlock. We showed that the choice of the balanced-byte position is very sensitive when the subkey relations are considered. As a result, we achieved the 22-round attack with (Data, Time, Memory) = $(2^{61}, 2^{70.00}, 2^{63})$, which is the current best attack for reduced-round LBlock in the single-key setting. Because designing light-weight cryptographic primitives is an actively discussed topic, we hope that this paper returns some useful feedback to future designs.

References

- [1] D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim, and S. Chee, "HIGHT: A new block cipher suitable for low-resource device," CHES 2006, ed. L. Goubin and M. Matsui, Lect. Notes Comput. Sci., vol.4249, pp.46–59, Springer-Verlag, Berlin, Heidelberg, New York, 2006.
- [2] ISO/IEC 18033-3:2010, Information technology — Security techniques — Encryption Algorithms — Part 3: Block ciphers, 2010.
- [3] A. Bogdanov, L.R. Knudsen, G. Leander, C. Paar, A. Poschmann, M.J.B. Robshaw, Y. Seurin, and C. Viskellsoe, "PRESENT: An ultra-lightweight block cipher," Cryptographic Hardware and Embedded Systems — CHES 2007, ed. P. Paillier and I. Verbauwhede, Lect. Notes Comput. Sci., vol.4727, pp.450–466, Springer-Verlag, Berlin, Heidelberg, New York, 2007.
- [4] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-bit blockcipher CLEFIA (extended abstract)," Fast Software Encryption — 14th International Workshop, FSE 2007, ed. A. Biryukov, Lect. Notes Comput. Sci., vol.4593, pp.181–195, Springer-Verlag, Berlin, Heidelberg, New York, 2007.
- [5] ISO/IEC 29192-2:2011, Information technology — Security techniques — Lightweight cryptography — Part 2: Block ciphers, 2011.
- [6] W. Wu and L. Zhang, "LBlock: A lightweight block cipher," ACNS 2011, ed. J. Lopez and G. Tsudik, Lect. Notes Comput. Sci., vol.6715, pp.327–344, Springer-Verlag, Berlin, Heidelberg, New York, 2011.
- [7] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: An ultra-lightweight blockcipher," Cryptographic Hardware and Embedded Systems — CHES 2011, ed. B. Preneel and T. Takagi, Lect. Notes Comput. Sci., vol.6917, pp.342–357, Springer-Verlag, Berlin, Heidelberg, New York, 2011.
- [8] J. Guo, T. Peyrin, A. Poschmann, and M.J.B. Robshaw, "The LED block cipher," Cryptographic Hardware and Embedded Systems — CHES 2011, ed. B. Preneel and T. Takagi, Lect. Notes Comput. Sci., vol.6917, pp.326–341, Springer-Verlag, Berlin, Heidelberg, New York, 2011.
- [9] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "TWINE: A lightweight block cipher for multiple platforms," Selected Areas in Cryptography SAC 2012, ed. L.R. Knudsen and H. Wu, Lect. Notes Comput. Sci., vol.7707, pp.339–354, Springer-Verlag, Berlin, Heidelberg, New York, 2012.
- [10] J. Daemen, L.R. Knudsen, and V. Rijmen, "The block cipher Square," Fast Software Encryption 1997, ed. E. Biham, Lect. Notes Comput. Sci., vol.1267, pp.149–165, Springer-Verlag, Berlin, Heidelberg, New York, 1997.
- [11] L.R. Knudsen and D. Wagner, "Integral cryptanalysis," FSE 2002, ed. J. Daemen and V. Rijmen, Lect. Notes Comput. Sci., vol.2365, pp.112–127, Springer-Verlag, Berlin, Heidelberg, New York, 2002.
- [12] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved cryptanalysis of Rijndael," FSE 2000, ed. B. Schneier, Lect. Notes Comput. Sci., vol.1978, pp.213–230, Berlin, Heidelberg, New York, 2000.
- [13] Y. Sasaki and L. Wang, "Meet-in-the-middle technique for integral attacks against feistel ciphers," Selected Areas in Cryptography SAC 2012, ed. L.R. Knudsen and H. Wu, Lect. Notes Comput. Sci., vol.7707, pp.234–251, Springer-Verlag, Berlin, Heidelberg, New York, 2012.
- [14] Y. Liu, D. Gu, Z. Liu, and W. Li, "Impossible differential attacks on reduced-round LBlock," ISPEC 2012, ed. M.D. Ryan, B. Smyth, and G. Wang, Lect. Notes Comput. Sci., vol.7232, pp.97–108, Springer-Verlag, Berlin, Heidelberg, New York, 2012.
- [15] F. Karakoç, H. Demirci, and A. Harmanci, "Impossible differential cryptanalysis of reduced-round LBlock," WISTP 2012, ed. I.G. Askoxylakis, H.C. Pöhls, and J. Posegga, Lect. Notes Comput. Sci., vol.7322, pp.179–88, Springer-Verlag, Berlin, Heidelberg, New York, 2012.
- [16] M. Minier and M. Naya-Plasencia, "A related key impossible differential attack against 22 rounds of the lightweight block cipher LBlock," Inf. Process. Lett., vol.112, no.16, pp.624–629, 2012.
- [17] S. Liu, Z. Gong, and L. Wang, "Improved related-key differential attacks on reduced round LBlock," ICICS 2012, ed. S.H. Qing and S.M. Yiu, Lect. Notes Comput. Sci., vol.7618, pp.58–69, Springer-Verlag, Berlin, Heidelberg, New York, 2012.
- [18] X.Y.Y. Wang, W. Wu, and L. Zhang, "Security on LBlock against biclique cryptanalysis," WISA 2012, ed. D.H. Lee and M. Yung, Lect. Notes Comput. Sci., vol.7690, pp.1–4, Springer-Verlag, 2012.
- [19] Y. Li, Integral cryptanalysis on block ciphers (in Chinese), [D], Institute of Software, Chinese Academy of Sciences, Beijing, 2012.
- [20] T. Suzaki and K. Minematsu, "Improving the generalized Feistel," Fast Software Encryption 2010, ed. S. Hong and T. Iwata, Lect. Notes Comput. Sci., vol.6147, pp.19–39, Springer-Verlag, Berlin, Heidelberg, New York, 2010.
- [21] H. Soleimany and K. Nyberg, "Zero-correlation linear cryptanalysis of reduced-round LBlock," Cryptology ePrint Archive, Report 2012/570, 2012. <http://eprint.iacr.org/2012/570>

Table A·1 Summary of the computation of $\bigoplus Z_{15}[4]$ in 22-round attack.

Step	Guessed key	#guessed bits (total)	Computed value	Discarded values	#texts	Values in the set	Complexity
2		0			2^{60}	$X_{22}^L[0, 1, 2, 4, 5, 6, 7], X_{22}^R[0, 1, 2, 3, 4, 5, 6, 7]$	
3	$K_{21}[0]$	4	$X_{21}^R[0]$	$X_{22}^L[2], X_{22}^R[0]$	2^{56}	$X_{22}^L[0, 1, 4, 5, 6, 7], X_{22}^R[1, 2, 3, 4, 5, 6, 7], X_{21}^R[0]$	$2^{60+4} = 2^{64}$
4	$K_{21}[4]$	8	$X_{21}^R[4]$	$X_{22}^L[6], X_{22}^R[4]$	2^{52}	$X_{22}^L[0, 1, 4, 5, 7], X_{22}^R[1, 2, 3, 5, 6, 7], X_{21}^R[0, 4]$	$2^{56+8} = 2^{64}$
5	$K_{21}[6]$	12	$X_{21}^R[5]$	$X_{22}^L[7], X_{22}^R[6]$	2^{48}	$X_{22}^L[0, 1, 4, 5], X_{22}^R[1, 2, 3, 5, 7], X_{21}^R[0, 4, 5]$	$2^{52+12} = 2^{64}$
6	$K_{20}[0]$	16	$X_{20}^R[0]$	$X_{22}^L[2], X_{21}^R[0]$	2^{44}	$X_{22}^L[0, 1, 4, 5], X_{22}^R[1, 3, 5, 7], X_{21}^R[4, 5], X_{20}^R[0]$	$2^{48+16} = 2^{64}$
7	$K_{21}[7]$	17	$X_{21}^R[3]$	$X_{22}^L[5]$	2^{44}	$X_{22}^L[0, 1, 4], X_{22}^R[1, 3, 5, 7], X_{21}^R[3, 4, 5], X_{20}^R[0]$	$2^{44+17} = 2^{61}$
8	$K_{21}[1]$	21	$X_{21}^R[6]$	$X_{22}^L[0]$	2^{44}	$X_{22}^L[1, 4], X_{22}^R[1, 3, 5, 7], X_{21}^R[3, 4, 5, 6], X_{20}^R[0]$	$2^{44+21} = 2^{65}$
9	$K_{20}[3]$	25	$X_{20}^R[7]$	$X_{22}^R[1], X_{21}^R[3]$	2^{40}	$X_{22}^L[1, 4], X_{22}^R[3, 5, 7], X_{21}^R[4, 5, 6], X_{20}^R[0, 7]$	$2^{44+25} = 2^{69}$
10	$K_{19}[7]$	27	$X_{19}^R[3]$	$X_{21}^R[5], X_{20}^R[7]$	2^{36}	$X_{22}^L[1, 4], X_{22}^R[3, 5, 7], X_{21}^R[4, 6], X_{20}^R[0], X_{19}^R[3]$	$2^{40+27} = 2^{67}$
11	$K_{20}[6]$	31	$X_{20}^R[5]$	$X_{22}^L[7], X_{21}^R[6]$	2^{32}	$X_{22}^L[1, 4], X_{22}^R[3, 5], X_{21}^R[4], X_{20}^R[0, 5], X_{19}^R[3]$	$2^{36+31} = 2^{67}$
12	$K_{19}[5]$	33	$X_{19}^R[2]$	$X_{21}^R[4], X_{20}^R[5]$	2^{28}	$X_{22}^L[1, 4], X_{22}^R[3, 5], X_{20}^R[0], X_{19}^R[2, 3]$	$2^{32+33} = 2^{65}$
13	$K_{21}[3]$	37	$X_{21}^R[7]$	$X_{22}^L[1]$	2^{28}	$X_{22}^L[4], X_{22}^R[3, 5], X_{21}^R[7], X_{20}^R[0], X_{19}^R[2, 3]$	$2^{28+37} = 2^{65}$
14	$K_{21}[5]$	41	$X_{21}^R[2]$	$X_{22}^L[4]$	2^{28}	$X_{22}^R[3, 5], X_{21}^R[2, 7], X_{20}^R[0], X_{19}^R[2, 3]$	$2^{28+41} = 2^{69}$
15	$K_{20}[7]$	45	$X_{20}^R[3]$	$X_{22}^R[5], X_{21}^R[7]$	2^{24}	$X_{22}^R[3], X_{21}^R[2], X_{20}^R[0, 3], X_{19}^R[2, 3]$	$2^{28+45} = 2^{73}$
16	$K_{18}[2]$	45	$X_{18}^R[1]$	$X_{20}^R[3], X_{19}^R[2]$	2^{20}	$X_{22}^R[3], X_{21}^R[2], X_{20}^R[0], X_{19}^R[3], X_{18}^R[1]$	$2^{24+45} = 2^{69}$
17	$K_{20}[2]$	49	$X_{20}^R[1]$	$X_{22}^R[3]$	2^{20}	$X_{21}^R[2], X_{20}^R[0, 1], X_{19}^R[3], X_{18}^R[1]$	$2^{20+49} = 2^{69}$
18	$K_{18}[3]$	49	$X_{18}^R[7]$	$X_{20}^R[1], X_{19}^R[3]$	2^{16}	$X_{21}^R[2], X_{20}^R[0], X_{18}^R[1, 7]$	$2^{20+49} = 2^{69}$
19	$K_{19}[0]$	50	$X_{19}^R[0]$	$X_{21}^R[2], X_{20}^R[0]$	2^{12}	$X_{19}^R[0], X_{18}^R[1, 7]$	$2^{16+50} = 2^{66}$
20	$K_{17}[1]$	50	$X_{17}^R[6]$	$X_{19}^R[0], X_{18}^R[1]$	2^8	$X_{18}^R[7], X_{17}^R[6]$	$2^{12+50} = 2^{62}$
21	$K_{16}[6]$	52	$X_{16}^R[5]$	$X_{18}^R[7], X_{17}^R[6]$	2^4	$X_{16}^R[5]$	$2^{8+52} = 2^{60}$
22	$K_{15}[5]$	56	$Z_{15}[4]$	$X_{16}^R[5]$	2^0	$\bigoplus Z_{15}[4]$	$2^{4+56} = 2^{60}$

Appendix A: Summary of Partial-Sum Applications for 22-Round Attack

The summary of the partial-sum technique and the complexity for each step is given in Table A·1.

Appendix B: Data for 21-Round Attack

In this section, we show the detailed data for the 21-round attack. Because the 21-round attack is not the best with respect to the number of attacked rounds, we only show the data and briefly summarize the attack. As shown in Table 2, we use the 6th byte of Z_{15}^R as the target balanced-byte.

B.1 6-Round Key-Recovery Phase for 21-Round Attack

Details of the 6-round key-recovery phase are shown in Fig. A·1. 13 key bytes and 12 ciphertext bytes related to the computation of $Z_{15}[0]$ are denoted by numbers in black square brackets. Similarly, 7 key bytes and 8 ciphertext bytes related to the computation of $X_{16}^L[0]$ are denoted by numbers in grey round brackets. The sum of $Z_{15}[0]$ and $X_{16}^L[0]$ are computed independently, and the bottle-neck is the computation for $Z_{15}[0]$.

B.2 Partial-Sum Applications for 21-Round Attack

The summary of the partial-sum technique and the complexity for each step is given in Table A·2. In the end, the complexity to compute $\bigoplus Z_{15}[0]$ is

Fig. A·1 21-round attack on LBlock.

$$2^{52} + 2^{52} + 2^{52} + 2^{52} + 2^{52} + 2^{52} + 2^{52} + 2^{52} + 2^{54} + 2^{54} + 2^{50} + 2^{46} + 2^{46} \approx 2^{56.60} \text{ 1-round computations.}$$

After the analysis, we obtain a list $L_{Z_{15}}$ with 2^{42} entries containing 46-bit information; $\bigoplus Z_{15}[6]$ and 42-bit key values.

Table A·2 Summary of the computation of $\bigoplus Z_{15}[0]$ in 21-round attack.

Step	Guessed key	#guessed bits (total)	Computed value	Discarded values	#texts	Values in the set	Complexity
2		0			2^{48}	$X_{21}^L[1, 3, 5, 6, 7], X_{21}^R[0, 1, 2, 3, 4, 6, 7]$	
3	$K_{20}[2]$	4	$X_{20}^R[1]$	$X_{21}^L[3], X_{21}^R[2]$	2^{44}	$X_{21}^L[1, 5, 6, 7], X_{21}^R[0, 1, 3, 4, 6, 7], X_{20}^R[1]$	$2^{48+4} = 2^{52}$
4	$K_{20}[3]$	8	$X_{20}^R[7]$	$X_{21}^L[1], X_{21}^R[3]$	2^{40}	$X_{21}^L[5, 6, 7], X_{21}^R[0, 1, 4, 6, 7], X_{20}^R[1, 7]$	$2^{44+8} = 2^{52}$
5	$K_{20}[4]$	12	$X_{20}^R[4]$	$X_{21}^L[6], X_{21}^R[4]$	2^{36}	$X_{21}^L[5, 7], X_{21}^R[0, 1, 6, 7], X_{20}^R[1, 4, 7]$	$2^{40+12} = 2^{52}$
6	$K_{20}[7]$	16	$X_{20}^R[3]$	$X_{21}^L[5], X_{21}^R[7]$	2^{32}	$X_{21}^L[7], X_{21}^R[0, 1, 6], X_{20}^R[1, 3, 4, 7]$	$2^{36+16} = 2^{52}$
7	$K_{19}[1]$	20	$X_{19}^R[6]$	$X_{21}^R[0], X_{20}^R[1]$	2^{28}	$X_{21}^L[7], X_{21}^R[1, 6], X_{20}^R[3, 4, 7], X_{19}^R[6]$	$2^{32+20} = 2^{52}$
8	$K_{19}[3]$	24	$X_{19}^R[7]$	$X_{21}^R[1], X_{20}^R[3]$	2^{24}	$X_{21}^L[7], X_{21}^R[6], X_{20}^R[4, 7], X_{19}^R[6, 7]$	$2^{28+24} = 2^{52}$
9	$K_{18}[6]$	28	$X_{18}^R[5]$	$X_{20}^R[7], X_{19}^R[6]$	2^{20}	$X_{21}^L[7], X_{21}^R[6], X_{20}^R[4], X_{19}^R[7], X_{18}^R[5]$	$2^{24+28} = 2^{52}$
10	$K_{20}[6]$	32	$X_{20}^R[5]$	$X_{21}^L[7]$	2^{20}	$X_{21}^R[6], X_{20}^R[4, 5], X_{19}^R[7], X_{18}^R[5]$	$2^{20+32} = 2^{52}$
11	$K_{18}[7]$	34	$X_{18}^R[3]$	$X_{20}^R[5], X_{19}^R[7]$	2^{16}	$X_{21}^R[6], X_{20}^R[4], X_{18}^R[3, 5]$	$2^{20+34} = 2^{54}$
12	$K_{19}[4]$	38	$X_{19}^R[4]$	$X_{21}^R[6], X_{20}^R[4]$	2^{12}	$X_{19}^R[4], X_{18}^R[3, 5]$	$2^{16+38} = 2^{54}$
13	$K_{17}[5]$	38	$X_{17}^R[2]$	$X_{19}^R[4], X_{18}^R[5]$	2^8	$X_{18}^R[3], X_{17}^R[2]$	$2^{12+38} = 2^{50}$
14	$K_{16}[2]$	38	$X_{16}^R[1]$	$X_{18}^R[3], X_{17}^R[2]$	2^4	$X_{16}^R[1]$	$2^{8+38} = 2^{46}$
15	$K_{15}[1]$	42	$Z_{15}[0]$	$X_{16}^R[1]$	2^0	$\bigoplus Z_{15}[0]$	$2^{4+42} = 2^{46}$

Fig. A·2 κ_{18} for computing $Z_{15}[0]$ and $X_{16}^L[0]$. Known bits are in grey.

B.3 Complexity Evaluation

We confirmed that $\bigoplus X_{16}^L[0]$ is computed with $7 \cdot 2^{36} \approx 2^{38.81}$ 1-round computations. We store $\bigoplus X_{16}^L[0]$ together with the 28-bit guessed keys in a table $L_{X_{16}^L}$. After the analysis, we obtain a list $L_{X_{16}^L}$ with 2^{28} entries.

After we make two lists $L_{Z_{15}}$ and $L_{X_{16}^L}$, we identify the right key candidates by checking the match. Besides the 4-bit condition $\bigoplus Z_{15}[0] = \bigoplus X_{16}^L[0]$, 11 key bits are overlapped between κ_{18} for computing $Z_{15}[0]$ and $X_{16}^L[0]$ as shown in Fig. A·2. Therefore, the key space can be reduced by $4 + 11 = 15$ bits in total. Note that 27 bits of κ_{18} are not included in both parts.

In summary, we analyze 3 sets of 2^{60} texts. The key space for the guessed 53 bits of κ_{18} is reduced into $2^{42+28-(15 \cdot 3)} = 2^{25}$. These 2^{25} and the other 27 bits are exhaustively searched with $2^{25+27} = 2^{52}$ LBlock computations. The data complexity is $3 \cdot 2^{60} \approx 2^{61.6}$ plaintexts. The time complexity is $3 \cdot (2^{56.60} + 2^{38.81})$ round functions and 2^{52} 21-round LBlock computations, which is approximately $2^{54.16}$ 21-round LBlock computations. The memory complexity is $12 \cdot 2^{48} \approx 2^{51.58}$ bytes to start the analysis.

Yu Sasaki received the B.E., M.E. and Ph.D. from The University of Electro-Communications in 2005, 2007, and 2010. Since 2007, he has been a researcher at NTT Secure Platform Laboratories. His current research interests are in cryptography. He was awarded a paper prize from SCIS 2007 and IEICE Trans. in 2009. He also received a best paper award from IWSEC 2009, SECRCRYPT 2012, and IWSEC 2012.

Lei Wang received the M.E. and Ph.D. from The University of Electro-Communications in 2009, and 2011, respectively. His current research interests are in cryptography. He was awarded a paper prize from SCIS 2008 and IEICE Trans. in 2009. He also received a best paper award from IWSEC 2009 and IWSEC 2012.