This document is downloaded from DR-NTU (https://dr.ntu.edu.sg)
Nanyang Technological University, Singapore.

Bitwise partial-sum : a new tool for integral analysis against ARX designs

Sasaki, Yu; Wang, Lei

2015

Sasaki, Y., & Wang, L. (2015). Bitwise partial-sum : a new tool for integral analysis against ARX designs. IEICE transactions on fundamentals of electronics, communications and computer sciences, E98.A(1), 49-60.

https://hdl.handle.net/10356/107090

https://doi.org/10.1587/transfun.E98.A.49

© 2015 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.E98.A.49]. 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.

Downloaded on 04 Apr 2024 22:53:20 SGT

PAPER *Special Section on Cryptography and Information Security*

Bitwise Partial-Sum: A New Tool for Integral Analysis against ARX Designs*

Yu SASAKI[†a)], ***Member and*** **Lei WANG**[††b)], ***Nonmember***

SUMMARY In this paper, we present a new cryptanalytic tool that can reduce the complexity of integral analysis against Addition-Rotation-XOR (ARX) based designs. Our technique is based on the partial-sum technique proposed by Ferguson et al. at FSE 2000, which guesses subkeys byte to byte in turn, and the data to be analyzed is compressed for each key guess. In this paper, the technique is extended to ARX based designs. Subkeys are guessed bit by bit, and the data is compressed with respect to the value of the guessed bit position and carry values to the next bit position. We call the technique *bitwise partial-sum*. We demonstrate this technique by applying it to reduced-round versions of HIGHT, which is one of the ISO standard 64-bit block ciphers. Another contribution of this paper is an independent improvement specific to HIGHT. By exploiting linear computations inside the round function, the number of guessed bits during the key recovery phase can be greatly reduced. Together with the bitwise partial-sum, the integral analysis on HIGHT is extended from previous 22 rounds to 26 rounds, while full HIGHT consists of 32 rounds.

key words: *integral analysis, partial-sum, bitwise partial-sum, ARX, HIGHT*

1. Introduction

A block cipher E takes a key and an input x, and returns an output y. For each key, block cipher must be an efficient permutation, and has an efficient inversion E^{-1}. The block cipher is a central primitive in modern cryptography, and is widely utilized in data encryption and data authentication. Therefore, cryptanalysts are devoted to evaluate the security of block ciphers. So far many cryptanalysis techniques have been devised. Among these techniques, one is named as *integral analysis*.

Integral analysis was firstly proposed by Daemen et al. to evaluate the security of Square cipher [4], and was later unified as integral analysis by Knudsen and Wagner [9]. It consists of two phases; an *integral distinguisher construction* and a *key recovery*. Let N be a block size of the target block cipher. In the first phase, the attacker aims to find a set of 2^t plaintexts for some $t < N$ such that the corresponding internal state value after a few encryption rounds have a certain property for any key e.g. the XOR sum of all internal state values in the set is 0 with a probability 1 for a fraction of bits of the state. This property is often called *balanced*. The concept of the integral distinguisher is depicted in Fig. 1. For the second phase, the attacker appends a few rounds to the end of the distinguisher, and aims to recover a part of subkeys in these rounds. The attacker obtains the corresponding ciphertexts of the set of chosen plaintexts by accessing the encryption oracle. Then, he guesses a part of subkeys and performs the partial decryption up to the output state of the integral distinguisher. The concept of the key recovery part is also depicted in Fig. 1. If the guess is correct, the XOR sum of the results always becomes 0. Otherwise, the XOR sum of the results becomes random. Hence, the key space can be reduced.

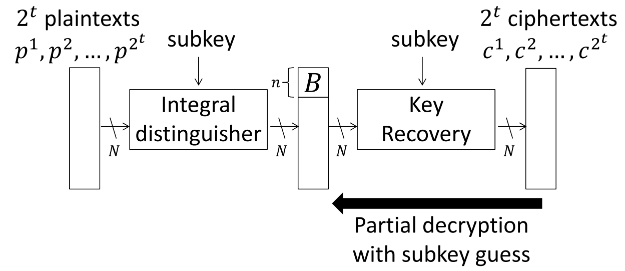

2^t plaintexts
$p^1, p^2, \ldots, p^{2^t}$
subkey
Integral distinguisher
N
n
B
subkey
Key Recovery
2^t ciphertexts
$c^1, c^2, \ldots, c^{2^t}$
Partial decryption with subkey guess

Fig. 1 Illustrations of integral distinguisher and key recovery. 2^t internal state values after a few rounds has a balanced property, denoted by B, for n bits of the state.

Cryptanalysts are continuously developing new techniques to enhance the integral analysis. Several results have been published on improving the integral distinguisher construction. Examples are multi-set analysis by Biryukov and Shamir [2], subword multi-set by Nakahara Jr. et al. [14], and bit-pattern based analysis by Z'aba et al. [21]. The analysis exploiting the property of the ARX based structure can be seen in saturation attack by Lucks [13] and tuple analysis by Aumasson et al. [1]. At the same time, techniques for the key recovery phase have been also getting improved, which is the main motivation of this paper. In order to make our contribution clear, we briefly illustrate two previous techniques of improving the key recovery phase below, which are related to this paper.

Ferguson et al. proposed a technique called *partial-sum* [5]. It reduces the complexity of the partial decryption up to the balanced state by guessing each subkey byte one after another. We use a toy example to show its procedure and significance. Suppose that $t = 24$ in Fig. 1, i.e. the attacker obtains 2^{24} ciphertexts c^i where $i = 1, 2, \ldots, 2^{24}$. Also sup-

Manuscript received March 25, 2014.
Manuscript revised July 31, 2014.

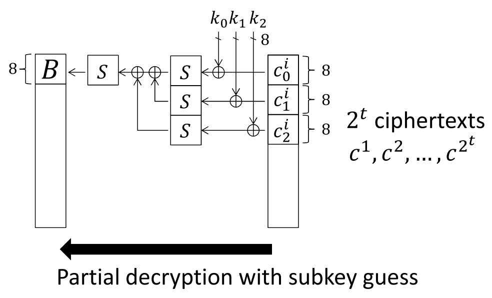
[†]The author is with NTT Secure Platform Laboratories, NTT Corporation, Musashino-shi, 180-8585 Japan.
[††]The author is with the Nanyang Technological University, Singapore.

[*]A preliminary version was presented in ICISC 2013. This is the full version. In particular, more detailed explanation of our attack on 26-round HIGHT is added.

a) E-mail: sasaki.yu@lab.ntt.co.jp
b) E-mail: wang.lei@ntu.edu.sg

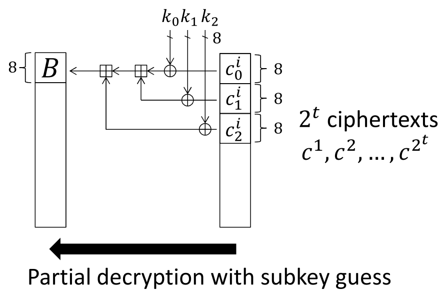
DOI: 10.1587/transfun.E98.A.49

Copyright © 2015 The Institute of Electronics, Information and Communication Engineers



$k_0 k_1 k_2$
8
B
S
c_0^i
c_1^i
c_2^i
2^t ciphertexts
$c^1, c^2, \ldots, c^{2^t}$
Partial decryption with subkey guess

Fig. 2 Partial decryption of the toy example.



$k_0 k_1 k_2$
8
B
c_0^i
c_1^i
c_2^i
2^t ciphertexts
$c^1, c^2, \ldots, c^{2^t}$
Partial decryption with subkey guess

Fig. 3 Partial decryption of the toy example with addition and XOR.

pose that the partial decryption up to the output state of the integral distinguisher is written as $S^{-1}(S^{-1}(c_0^i \oplus k_0) \oplus S^{-1}(c_1^i \oplus k_1) \oplus S^{-1}(c_2^i \oplus k_2))$, where (c_0^i, c_1^i, c_2^i) are 3 bytes (=24 bits) of the ciphertext c^i, k_0, k_1, k_2 are 3 bytes of subkeys, and $S(\cdot)$ is a byte-wise S-box application. The key recovery part with this example is depicted in Fig. 2. In order to recover those subkeys, the attacker exhaustively guesses 2^{24} possibilities of (k_0, k_1, k_2) and computes the XOR sum of the partial decryption for 2^{24} ciphertexts $c^1, c^2, \ldots, c^{2^{24}}$, i.e., the attacker computes the following equation:

$$\bigoplus_{i=1}^{2^{24}} S^{-1}(S^{-1}(c_0^i \oplus k_0) \oplus S^{-1}(c_1^i \oplus k_1) \oplus S^{-1}(c_2^i \oplus k_2)).$$

With a straightforward method, it takes $2^{24+24} = 2^{48}$ computations. With the partial-sum technique, the attacker can compute it only with 2^{40} computations. In details, each key byte is guessed in turn and the data to be analyzed is compressed for each key guess (before the rest of the key bytes are guessed). In the above example, at first k_0 and k_1 are guessed and the sum of the first two terms, i.e., $\bigoplus_{i=1}^{2^{24}} S^{-1}(c_0^i \oplus k_0) \oplus S^{-1}(c_1^i \oplus k_1)$ are computed. Let c' be this value. The computation of c' takes $2^{16+24} = 2^{40}$ computations. Then, 2^{24} data for 2-byte tuple (c', c_2) are generated. This causes many overlaps of the data; roughly 2^8 overlaps for each value of (c', c_2). Hence, the attacker only picks values that appear odd times. This reduces the data size into 2^{16}. Finally, the entire sum is computed by guessing k_2, which takes $2^{16} \cdot 2^8 \cdot 2^{16} = 2^{40}$ computations. This is faster than the straightforward method.

Sasaki and Wang introduced *meet-in-the-middle technique* for the key recovery phase of the integral analysis against block-ciphers with Feistel network [17]. It exploits the property that the balanced state is represented by an XOR of two variables. Then, the XOR sum of each variable are computed independently, and the attacker checks the match of their values like a meet-in-the-middle attack. It separates the partial decryption into two independent parts, and thus the complexity can be reduced.

Our contributions

In this paper, we extend the partial-sum technique to ARX designs, beyond the mere application to byte-oriented ciphers. We also use a toy example to illustrate our new tool. Suppose that inside the partial decryption in Fig. 2, non-linearity is introduced by the modular addition instead of the S-box application. More precisely, suppose that the partial decryption is written as $(c_0^i \oplus k_0) \boxplus (c_1^i \oplus k_1) \boxplus (c_2^i \oplus k_2)$, where the symbols "$\oplus$" and "$\boxplus$" represent the bitwise XOR and the addition on modulo 2^8, respectively. The key recovery part with this example is depicted in Fig. 3. The attacker aims to compute the XOR-sum for the exhaustive guess of (k_0, k_1, k_2), i.e.,

$$\bigoplus_{i=1}^{2^{24}} [(c_0^i \oplus k_0) \boxplus (c_1^i \oplus k_1) \boxplus (c_2^i \oplus k_2)]$$

This paper is motivated by the following question:

> *What is the best strategy to compute* $\bigoplus_{i=1}^{2^{24}} [(c_0^i \oplus k_0) \boxplus (c_1^i \oplus k_1) \boxplus (c_2^i \oplus k_2)]$*?*

The partial-sum technique [5] can be applied to compute this equation, i.e., two keys are guessed and the data is compressed. However, we observe that the computation can be much faster by guessing key values bit-by-bit and compressing the data for each guess. For example, let us consider the computation of the first two terms; $c' = (c_0 \oplus k_0) \boxplus (c_1 \oplus k_1)$. We guess two key bits, which are the LSB of k_0 and the LSB of k_1. Then, we can compute the LSB of c' and the carry bit to the second LSB. After this computation, 2-bit information, which are the LSB of c_0 and the LSB of c_1 are discarded. At this stage, we newly obtain 2-bit information and discard 2-bit information. Hence, no advantage is generated. We then guess two key bits, which are the second LSB of k_0 and the second LSB of k_1. Then, we can compute the second LSB of c' and the carry bit to the third LSB. After this computation, 3-bit information, which are the second LSB of c_0 and c_1 and the carry value to the second LSB are discarded. In this time, we newly obtain 2-bit information, but discarded 3-bit information. Hence, the data to be analyzed is compressed by 1 bit. With this approach, the complexity for the above equation is minimized. We call the technique *bitwise partial-sum*.

The bitwise partial-sum leads to more advantages. We focus on the computation for the MSB. As shown above, the bitwise partial-sum computes the carry value to the MSB when we analyze the second MSB. Therefore, the analysis on the MSB is completely linear because we do not need to compute the carry value from the MSB. Therefore, in the

Table 1 Comparison of attack results on HIGHT.

Model	Approach	#Rounds	Data	Time	Memory (bytes)	Reference
single-key	Integral	18	2^{62}	2^{36}	2^{20}	[20]
	Integral	22	2^{62}	$2^{118.71}$	2^{64}	[22]
	Integral	22	2^{62}	$2^{102.35}$	2^{64}	[17]
	Integral	22	$2^{62.15}$	$2^{67.28}$	2^{56}	This paper
	Integral	26	$2^{58.58}$	$2^{123.07}$	$2^{115.58}$	This paper
	Imp. Diff.	18	$2^{46.8}$	$2^{109.2}$	N/A	[7]
	Imp. Diff.	25	2^{60}	$2^{126.78}$	N/A	[12]
	Imp. Diff.	26	2^{61}	$2^{119.53}$	2^{109}	[16]
	Imp. Diff.	26	$2^{61.6}$	$2^{114.35}$	$2^{87.6}$	[3]
	Imp. Diff.	27	2^{58}	$2^{126.6}$	2^{120}	[3]
	Zero-Corr.	26	$2^{62.79}$	$2^{119.1}$	2^{43}	[19] †
	Zero-Corr.	27	$2^{62.79}$	$2^{120.78}$	2^{43}	[19] ‡
related-key	Rectangle	26	$2^{51.2}$	$2^{120.41}$	N/A	[12]
	Imp. Diff.	28	2^{60}	$2^{125.54}$	N/A	[12]
	Imp. Diff.	31	2^{64}	$2^{127.28}$	2^{117}	[16]
	Differential	32	$2^{57.84}$	$2^{125.83}$	N/A	[10]

†: Attacked rounds are from round 4 to round 29.
‡: Attacked rounds are from round 4 to round 30.

above equation, 3 bits for the MSBs of c_0, c_1, c_2 can be compressed into 1 bit of $c_0 \oplus c_1 \oplus c_2$ with respect to the MSB at the very beginning stage of the analysis. Moreover, for 3 bits of the MSBs of k_0, k_1, k_2, guessing only 1-bit information for their XOR relation is enough.

We stress that the bitwise partial-sum technique is different from the known techniques. The main goal of the bit-pattern based analysis [21], saturation attack [13] and tuple analysis [1] is extending an integral distinguisher by introducing new properties other than active, balanced, and constant, whereas the goal of the bitwise partial-sum is reducing the complexity of the key recovery phase by processing the partial decryption, and tracing the carry effect in bitwise. In fact, we use the same integral distinguisher as the previous work, but improve the complexity.

In this paper, we demonstrate the bitwise partial-sum technique for HIGHT [7], which was standardized by ISO as a 64-bit block-cipher [8]. As an independent improvement, we show an observation specific to HIGHT, which exploits linearity inside the round function. This reduces a number of guessed subkey bits during the key recovery phase. We present our attacks on HIGHT by combining these two techniques. Then, we successfully improve the previous 22-round attack on HIGHT, and moreover the number of attacked rounds is extended to 26 rounds. Although the best single-key attack on HIGHT breaks 27 rounds with an impossible differential attack [3], this is a significant improvement regarding the integral analysis including several new observations. The attack results are summarized in Table 1.

Paper outline

The paper is organized as follows. In Sect. 2, we introduce the previous techniques for integral analysis. In Sect. 3, we describe our main idea called bitwise partial-sum. In Sect. 4, we show another optimization specific for HIGHT, and apply it to 22- and 26-round HIGHT together with the bitwise partial-sum. Finally, we conclude the paper in Sect. 5.

2. Related Work

2.1 Integral Analysis and Improved Techniques

A brief description of integral attack has already been given in Sect. 1. To discuss integral distinguishers, we use the following notations to describe the property of each byte.

"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.

2.1.1 Partial-Sum

The partial-sum technique was introduced by Ferguson et al. [5] 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 integral analysis on AES, the partial decryption involves 5 key bytes and 4 ciphertext bytes. Suppose that the number of data to be analyzed is 2^{32} and the byte position b of the n-th ciphertext is denoted by $c_{b,n}$. Then, the equation is described as follows.

$$\bigoplus_{n=1}^{2^{32}} \Big[S_4\Big(S_0(c_{0,n} \oplus k_0) \oplus S_1(c_{1,n} \oplus k_1) \oplus S_2(c_{2,n} \oplus k_2) \oplus S_3(c_{3,n} \oplus k_3) \oplus k_4 \Big) \Big]. \tag{1}$$

With a straightforward method, the analysis exhaustively guesses 40-bit key values and performs the partial decryption for all 2^{32} texts. Therefore, it takes $2^{32+40} = 2^{72}$ partial decryptions. The partial-sum technique can perform this computation only with 2^{48} partial decryptions.

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 $\bigoplus_{p=0}^{i}(S_p(c_{p,n} \oplus k_p))$. 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

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

The original set includes 2^{32} texts, but now only 3-byte information (x_1, c_2, c_3) is included. 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 $2^{16} \times 2^{32} = 2^{48}$,

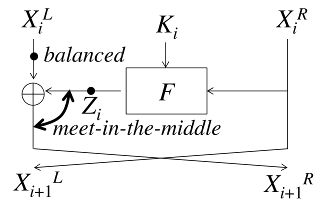


Fig. 4 Meet-in-the-middle techniques in [17].

for the guess of k_2 is $2^{16}\times 2^8\times 2^{24} = 2^{48}$, for the guess of k_3 is $2^{24}\times 2^8\times 2^{16} = 2^{48}$, for the guess of k_4 is $2^{32}\times 2^8\times 2^8 = 2^{48}$. In the end, the complexity is preserved to be 2^{48} until the last computation.

2.1.2 Meet-in-the-Middle Matching for the Key Recovery

Sasaki and Wang introduced *meet-in-the-middle technique* for the key recovery phase of the integral analysis against block-ciphers with Feistel network [17], which can also be seen in the analysis of TWINE [18].

Suppose that the input value to round i in a Feistel cipher is $X_i^L \| X_i^R$, and the output value $X_{i+1}^L \| X_{i+1}^R$ where $X_{i+1}^L = X_i^R$ and $X_{i+1}^R = X_i^L \oplus F(K_i, X_i^R)$, which is shown in Fig. 4. Also suppose that the balanced property is preserved in a variable X_i^L i.e., $\bigoplus X_i^L = 0$. Due to the linear relation $X_i^L = Z_i \oplus X_{i+1}^R$, where Z_i is the output of the round function for round i, the equation $\bigoplus X_i^L = 0$ can be written as $\bigoplus Z_i = \bigoplus X_{i+1}^R$. Therefore, the sum of each term can be computed independently, and the key values that result in the balanced state can be identified with the same manner as the meet-in-the-middle attack.

3. Bitwise Partial-Sum

In this section, we describe our new technique called *bitwise partial-sum*, which improves the complexity of the partial-sum technique for ARX designs. Suppose that an n-bit variable Z is computed with n-bit variables X, Y and n-bit unknown keys K, K'[†]. Also suppose that 2^{2n} pairs of (X, Y) is given to the attacker and the goal of the attacker is computing Z for the exhaustive guess of K and K'. As computations of Z, we consider the following four operations, which are also depicted in Fig. 5.

$$Z = (X \oplus K) \boxplus Y, \qquad Z = (X \boxplus K) \oplus Y,$$
$$Z = (X \oplus K) \boxplus (Y \oplus K'), \qquad Z = (X \boxplus K) \oplus (Y \boxplus K'),$$

We describe X in bitwise with $X^{n-1}\|X^{n-2}\|\cdots\|X^1\|X^0$. The similar notations are used for Y, Z, K, and K'. We denote the carry value to bit position i by p^i.

To compute Z in each of the above operations, in the previous work, the key values K (and K') are exhaustively guessed, and for each guess, the equation is computed for all 2^{2n} pairs of (X, Y). Therefore, the complexity is $2^{2n}\cdot 2^n = 2^{3n}$

[†]For HIGHT, the value of n is 8. Here, we describe the analysis in a general form.

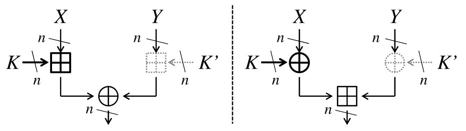


Fig. 5 Models of the analytic target.

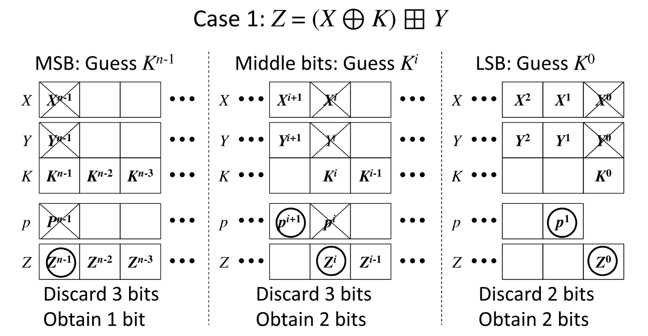


Fig. 6 Overview of the bitwise partial-sum for case 1. Each cell represents each bit. Crossed cells represent the discarded bits. Circles cells represent the newly obtained bits.

for the single-key cases, and $2^{2n}\cdot 2^{2n} = 2^{4n}$ for the two-key cases. The bitwise partial-sum can reduce the complexity of these computations to $n\cdot 2^{2n+1}$ for the single-key cases and 2^{3n+2} for the two-key cases by guessing keys bit by bit and computing Z bit by bit.

Single-key cases

We start with explaining the complexity to compute $Z = (X \oplus K) \boxplus Y$ and $Z = (X \boxplus K) \oplus Y$. The procedure and the complexity for two cases are almost the same. Hence, we only explain the case for $Z = (X \oplus K) \boxplus Y$ in details. The overview is shown in Fig. 6. The analysis starts from 2^{2n} texts of (X, Y). The procedure is divided into three parts; LSB, middle bits, and MSB.

LSB: Guess the 1-bit value K^0. For each of 2^{2n} texts, compute $(X^0 \oplus K^0) \boxplus Y^0$ to obtain 2-bit information Z^0 and p^1. After that 2-bit information X^0 and Y^0 is no longer used, and thus we can remove those 2-bit information for the further procedure. Hence, 2^{2n} texts of $(X^{n-1}\|\cdots\|X^0, Y^{n-1}\|\cdots\|Y^0)$ are updated to 2^{2n} texts of $(X^{n-1}\|\cdots\|X^1, Y^{n-1}\|\cdots\|Y^1, p^1, Z^0)$.

Middle bits (bit position i for $i = 1, 2, \ldots, n-2$): Guess the 1-bit value K^i. For each text, compute $(X^i \oplus K^i) \boxplus Y^i \boxplus p^i$ to obtain 2-bit information Z^i and p^{i+1} and then discard 3-bit information X^i, Y^i, p^i. Hence, $2^{2n-(i-1)}$ texts of $(X^{n-1}\|\cdots\|X^i$, $Y^{n-1}\|\cdots\|Y^i, p^i, Z^{i-1}\|\cdots\|Z^0)$ are updated to 2^{2n-i} texts of $(X^{n-1}\|\cdots\|X^{i+1}$, $Y^{n-1}\|\cdots\|Y^{i+1}$, p^{i+1}, $Z^i\|\cdots\|Z^0)$. Count how many times each tuple of $(X^{n-1}\|\cdots\|X^{i+1}$, $Y^{n-1}\|\cdots\|Y^{i+1}, p^{i+1}, Z^i\|\cdots\|Z^0)$ appears, and only pick the ones that appear odd times. The size of the texts will be reduced from $2^{2n-(i-1)}$ to 2^{2n-i}.

MSB: Guess the 1-bit value K^{n-1}. For each text, compute $(X^{n-1} \oplus K^{n-1} \oplus Y^{n-1} \oplus p^{n-1})$ to obtain 1-bit information Z^{n-1} and then discard 3-bit information $X^{n-1}, Y^{n-1}, p^{n-1}$. Hence, 2^{n+2} texts of $(X^{n-1}, Y^{n-1}, p^{n-1}, Z^{n-2}\|\cdots\|Z^0)$ are updated to 2^n texts of $(Z^{n-1}\|\cdots\|Z^0)$. Count how many times each tuple of $(Z^{n-1}\|\cdots\|Z^0)$ appears, and only pick the ones that appear odd times. The size of the texts will be reduced from 2^{n+2} to 2^n.

The complexity for the LSB (bit position 0) is $2 \cdot 2^{2n}$ XOR operations and $2 \cdot 2^{2n}$ addition operations, then 2^{2n} texts will remain. The complexity for bit position 1 is $2^2 \cdot 2^{2n}$ XOR operations and $2^2 \cdot 2^{2n}$ addition operations, then 2^{2n-1} texts will remain. The complexity for bit position 2 is $2^3 \cdot 2^{2n-1}$ XOR operations and $2^3 \cdot 2^{2n-1}$ addition operations, then 2^{2n-2} texts will remain. The complexity for bit position i where $i = 3, 4, \ldots, n-2$ is $2^{i+1} \cdot 2^{2n-(i-1)}$ XOR operations and $2^{i+1} \cdot 2^{2n-(i-1)}$ addition operations, then 2^{2n-i} texts will remain. The complexity for the MSB is $2^n \cdot 2^{n+2}$ XOR operations, and 2^n texts will remain. Because the complexity for each bit is about 2^{2n+2} XOR operations and addition operations, the total complexity is about $n \cdot 2^{2n+2}$ XOR operations and addition operation. This is faster than the previous analysis which requires 2^{3n} XOR and addition operations. For example, for $n = 8$, the previous analysis requires 2^{24} operations while the bitwise partial-sum requires 2^{21} operations. The advantage of the bitwise partial-sum becomes much bigger for a bigger n. For example, Three-fish block-cipher adopts a 64-bit ARX design. For $n = 64$, the previous analysis requires 2^{192} while the bitwise partial-sum only requires 2^{136} operations.

Optimization of the single-key case

The complexity can be further improved. We observe that the computation of the MSB is linear, and thus the MSBs of two variables X^{n-1} and Y^{n-1} are only used in the linear computation. Hence, at the very beginning of the procedure, we can compute $X^{n-1} \oplus Y^{n-1}$, and 2^{2n} texts of (X, Y) can be compressed into 2^{2n-1} texts of $(X^{n-1} \oplus Y^{n-1}, X^{n-2}\|\cdots\|X^0, Y^{n-2}\|\cdots\|Y^0)$ by counting how many times each tuple appears and only picking the ones that appear odd times. This halves the complexity, and thus the total complexity is about $n \cdot 2^{2n+1}$ XOR and addition operations.

Two-key cases

We explain the two-key cases, i.e., the complexity to compute $Z = (X \oplus K) \boxplus (Y \oplus K')$ and $Z = (X \boxplus K) \oplus (Y \boxplus K')$. The basic procedure for the two-key cases are the same as the one for the single-key case. First, we explain the case for $Z = (X \oplus K) \boxplus (Y \oplus K')$.

LSB: Guess the 2-bit values K^0 and K'^0. For each 2^{2n} texts, compute 2-bit information Z^0 and p^1 and then discard 2-bit information X^0 and Y^0. The text size after the analysis is 2^{2n}.

Middle bits (bit position i for $i = 1, 2, \ldots, n-2$): Guess the 2-bit values K^i and K'^i. For each texts, compute 2-bit information Z^i and p^{i+1} and then discard 3-bit information X^i, Y^i, p^i. Count how many times each tuple of $(X^{n-1}\|\cdots\|X^{i+1}, Y^{n-1}\|\cdots\|Y^{i+1}, p^{i+1}, Z^i\|\cdots\|Z^0)$ appears, and only pick the ones that appear odd times. The size of the texts will be reduced from $2^{2n-(i-1)}$ to 2^{2n-i}.

MSB: Guess the 2-bit values K^{n-1} and K'^{n-1}. For each texts, compute 1-bit information Z^{n-1} and then discard 3-bit information $X^{n-1}, Y^{n-1}, p^{n-1}$. Count how many times each tuple of $(Z^{n-1}\|\cdots\|Z^0)$ appears, and only pick the ones that appear odd times. The size of the texts will be reduced from 2^{n+2} to 2^n.

The complexity for the LSB (bit position 0) is $(2)^2 \cdot 2^{2n}$ operations, and 2^{2n} texts will remain. The complexity for bit position 1 is $(2^2)^2 \cdot 2^{2n}$ operations, and 2^{2n-1} texts will remain. The complexity for bit position 2 is $(2^3)^2 \cdot 2^{2n-1}$ operations, and 2^{2n-2} texts will remain. The complexity for bit position i where $i = 3, 4, \ldots, n-2$ is $(2^{i+1})^2 \cdot 2^{2n-(i-1)}$ operations, and 2^{2n-i} texts will remain. The complexity for the MSB is $(2^n)^2 \cdot 2^{n+2}$ operations, and 2^n texts will remain. Therefore, the total complexity is about

$$(2)^2 \cdot 2^{2n} + [(2^2)^2 \cdot 2^{2n} + (2^3)^2 \cdot 2^{2n-1} + \cdots + (2^{n-1})^2 \cdot 2^{n+3}] + (2^n)^2 \cdot 2^{n+2}.$$

The first term is smaller than 2^{2n+3}, thus the equation is smaller than

$$\begin{aligned}&2^{2n+3} + 2^{2n+4} + 2^{2n+5} + \cdots + 2^{2n+(n+1)} + 2^{2n+(n+2)}\\ &= 2^{2n+3}(1 + 2^1 + 2^2 + \cdots + 2^{n-1})\\ &< 2^{3n+3}.\end{aligned}$$

This is faster than the previous analysis which requires 2^{4n} XOR and addition operations.

Optimization of the two-key case

Regarding the MSBs of two variables X^{n-1} and Y^{n-1}, the same technique as the one for the single-key case can be exploited, namely, take the XOR of X^{n-1} and Y^{n-1} and compress the data by 1 bit at the very beginning of the analysis. This reduces the total complexity by 1 bit, and thus the total complexity becomes about 2^{3n+2} XOR and addition operations.

Moreover, the MSBs of two keys K^{n-1} and K'^{n-1} are only used in the linear operation. Therefore, instead of guessing these two key bits, guessing 1-bit relation of these bits, i.e., $K^{n-1} \oplus K'^{n-1}$, is enough. This reduces the total complexity by 1 bit, and thus the total complexity becomes about 2^{3n+1} XOR and addition operations.

Evaluation for $Z = (X \boxplus K) \oplus (Y \boxplus K')$

The complexity for $Z = (X \boxplus K) \oplus (Y \boxplus K')$ is a little bit worse than the complexity for $Z = (X \oplus K) \boxplus (Y \oplus K')$. This is because the equation $(X \boxplus K) \oplus (Y \boxplus K')$ contains

Table 2 Summary of the complexity of the bitwise partial-sum.

Target equation	Previous partial-sum	bitwise partial-sum
$Z = (X \oplus K) \boxplus Y$	2^{3n}	$n \cdot 2^{2n+1}$
$Z = (X \boxplus K) \oplus Y$	2^{3n}	$n \cdot 2^{2n+1}$
$Z = (X \oplus K) \boxplus (Y \oplus K')$	2^{4n}	2^{3n+1}
$Z = (X \boxplus K) \oplus (Y \boxplus K')$	2^{4n}	2^{3n+2}

The unit of the complexity is a single computation of n-bit XOR or n-bit modular addition.

two additions, and thus we need to store 2-bit carry values in the analysis of each bit. Compared to the case of $Z = (X \oplus K) \boxplus (Y \oplus K')$, the size of texts to be analyzed is doubled. This increases the final complexity after the optimization is applied from 2^{3n+1} to 2^{3n+2} XOR and addition operations.

Summary of the bitwise partial-sum

The comparison of the complexities to compute each of 4 equations with the previous method (bytewise partial-sum) and ours is given in Table 2. It indicates that the advantage of the bitwise partial-sum increases as n increases. In the next section, we apply the bitwise partial-sum to HIGHT, where the size of n is 8. The impact is relatively small because the advantage is at most a factor of 2^7. Some *ARX*-based block-ciphers adopt a bigger n, e.g., XTEA [15] adopts $n = 32$ and Threefish [6] adopts $n = 64$. In such cases, the impact becomes much bigger.

4. Improved Integral Analysis on HIGHT

In this section, we improve the integral analysis on HIGHT by using the bitwise partial-sum technique. The section starts from introducing HIGHT specification (Sect. 4.1) and the previous integral analysis on 22 rounds (Sect. 4.2). Next, we propose a new observation specific to the HIGHT round function (Sect. 4.3). Then, by combining all techniques, we improve the complexity of the 22-round attack (Sect. 4.4). Finally, we extend the number of attacked rounds as much as possible (Sect. 4.5).

4.1 Specification of HIGHT

HIGHT is a light-weight block-cipher proposed at CHES 2006 by Hong et al. [7]. The block size is 64 bits and the key size is 128 bits. It adopts the type-2 generalized Feistel structure with 8 branches and 32 rounds, and the round function consists of the ARX structure.

The plaintext is loaded into an internal state $X_{0,7} \| X_{0,6} \| \cdots \| X_{0,0}$ where the size of each $X_{i,j}$ is 8 bits. At first, $X_{0,7} \| X_{0,6} \| \cdots \| X_{0,0}$ is updated by the pre-whitening operation. We omit its details because the pre-whitening does not impact to our attack. Then, the state $X_{i,7} \| X_{i,6} \| \cdots \| X_{i,0}$ is updated by the following operation. For $i = 0, 1, \ldots, 31$

$$X_{i+1,0} = X_{i,7} \oplus (F_0(X_{i,6}) \boxplus SK_{4i+3}), \qquad X_{i+1,1} = X_{i,0},$$
$$X_{i+1,2} = X_{i,1} \boxplus (F_1(X_{i,0}) \oplus SK_{4i}), \qquad X_{i+1,3} = X_{i,2},$$
$$X_{i+1,4} = X_{i,3} \oplus (F_0(X_{i,2}) \boxplus SK_{4i+1}), \qquad X_{i+1,5} = X_{i,4},$$

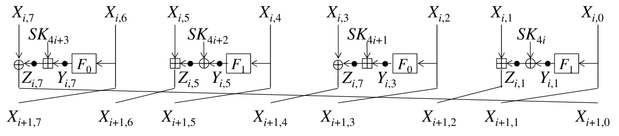


Fig. 7 Round function.

$$X_{i+1,6} = X_{i,5} \boxplus (F_1(X_{i,4}) \oplus SK_{4i+2}), \qquad X_{i+1,7} = X_{i,6},$$

where $F_0(x) = (x \lll 1) \oplus (x \lll 2) \oplus (x \lll 7)$, $F_1(x) = (x \lll 3) \oplus (x \lll 4) \oplus (x \lll 6)$, and "$\lll s$" denotes the s-bit left rotation. The swap of the byte positions is not executed in the last round. The round function is depicted in Fig. 7. We denote the internal state between F and the key addition by $Y_{i,1}, Y_{i,3}, Y_{i,5}, Y_{i,7}$ and the internal state right after the key addition by $Z_{i,1}, Z_{i,3}, Z_{i,5}, Z_{i,7}$. Finally, the post-whitening operation is performed as follows.

$$X_{32,0} \leftarrow X_{32,0} \boxplus WK_4, \qquad X_{32,2} \leftarrow X_{32,2} \oplus WK_5,$$
$$X_{32,4} \leftarrow X_{32,4} \boxplus WK_6, \qquad X_{32,6} \leftarrow X_{32,6} \oplus WK_7.$$

$X_{32,7} \| X_{32,6} \| \cdots \| X_{32,0}$ are output as the ciphertext. Note that a figure of the round function in the CHES 2006 version [7] shows an incorrect specification, which the order of subkeys in round i is different from the above description. Moreover, the previous integral analysis on HIGHT [17], [22] attacked the incorrect round function though they can be converted to the correct one as well. In 2009, the designers showed a figure of the round function with the correct subkey order [11]. Our analysis is targeting the correct one, hence the subkey order is different from the previous work [17], [22].

Subkeys and whitening keys consist of a part of the master key K and a constant value. The master key K is represented as $K_{15} \| K_{14} \| \cdots \| K_0$, where the size of each K_i is 8 bits. The post-whitening keys WK_i where $i = 4, 5, 6, 7$ are K_{i-4}. 128 subkeys are generated by the following algorithm with 128 constant values denoted by δ_i.

```
1. for $i$ = 1 to 7
2.         for $j$ = 1 to 7
3.                 $SK_{16 \cdot i+j} = K_{j-1 \bmod 8} \boxplus \delta_{16 \cdot i+j}$.
4.         for $j$ = 1 to 7
5.                 $SK_{16 \cdot i+j+8} = K_{(j-1 \bmod 8)+8} \boxplus \delta_{16 \cdot i+j+8}$.
```

We denote the k-th bit of a byte $X_{i,j}$ by $X^k_{i,j}$.

4.2 Previous Integral Analysis on 22-Round HIGHT

Zhang et al. presented a 17-round integral distinguisher on HIGHT [22]. For a set of 2^{56} plaintexts with the form of (A, A, A, A, A, A, A, C), the state after 17 rounds, $(X_{17,7} \| X_{17,6} \| \cdots \| X_{17,0})$, has the form of $(?, ?, ?, ?, B^0, ?, ?, ?)$, where B^0 stands for the balanced state with respect to the 0-th bit. By appending 5 rounds after this distinguisher, Zhang et al. showed a 22-round key recovery attack with a complexity of 2^{62} chosen plaintexts, $2^{118.71}$ 22-round HIGHT computations and a memory to store 2^{64} bytes.

The key recovery phase in [22] was later improved

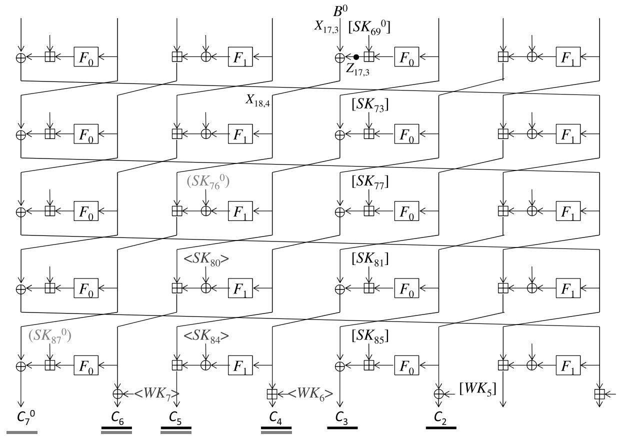


Fig. 8 Key recovery phase for 22-round HIGHT.

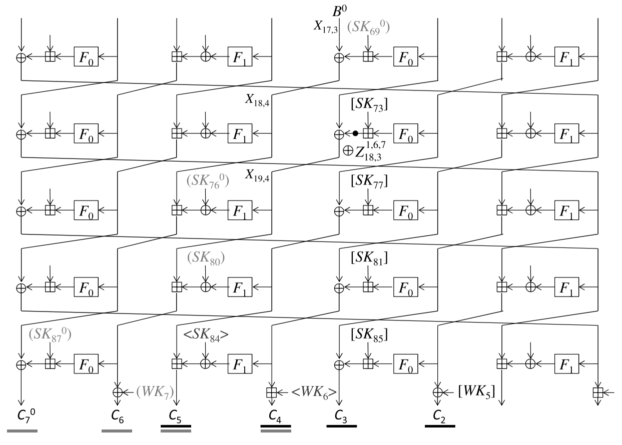


Fig. 9 Improved key recovery phase with exploiting more linearity.

by Sasaki and Wang [17]. They applied the meet-in-the-middle technique to reduce the complexity. The condition for $\bigoplus X^0_{17,3} = 0$ is written as

$$\bigoplus X^0_{18,4} = \bigoplus Z^0_{17,3}. \tag{2}$$

Their key recovery phase is illustrated in Fig. 8. The characters with round parenthesis describe the partial decryption for $Z^0_{17,3}$ that involves 73 bits of subkeys and 40 bits of ciphertexts. The characters with square parenthesis describe the partial decryption for $X^0_{18,4}$ that involves 34 bits of subkeys and 25 bits of ciphertexts. The characters with angle parenthesis describe the overlapped part. If the key schedule is considered, the numbers of subkey bits to compute $Z^0_{17,3}$ and $X^0_{18,4}$ are 64 and 26 respectively. The dominant complexity is the computations for $\bigoplus Z^0_{17,3}$ (with round parenthesis) that requires $2^{40+64} = 2^{104}$ partial decryptions to analyze a single set.

To reduce the key space into 1, the attack is iterated 65 times. With some optimization, they perform this part with about 2^{106} partial decryptions. The final complexity was evaluated by considering the ratio of F_0/F_1 functions to be computed, which is $2^{106} \times 7/88 \approx 2^{102.35}$ 22-round HIGHT encryptions.

4.3 Exploiting Linearly for Optimizing Matching Position

Recall Eq. (2), which is the application of the meet-in-the-middle technique by [17]. It is based on the fact that the balanced bit $X^0_{17,3}$ can be written as a linear combination of two variables $X^0_{18,4}$ and $Z^0_{17,3}$. We show that we can extend this concept by exploiting the linear computations inside the round function. As is explained in [17], the complexity for computing $\bigoplus Z^0_{17,3}$ is always much bigger than the one for $\bigoplus X^0_{18,4}$. Therefore, we aim to reduce the number of subkeys involved in the computation of $\bigoplus Z^0_{17,3}$.

We observe that $Z^0_{17,3}$ is computed by $SK^0_{69} \boxplus Y^0_{17,3}$. Because we only focus on the LSB, this computation is linear, i.e., $Z^0_{17,3} = SK^0_{69} \oplus Y^0_{17,3}$. Therefore, SK^0_{69} can be moved to the computation of $\bigoplus X^0_{18,4}$, namely $\bigoplus(X^0_{18,4} \oplus SK^0_{69}) = \bigoplus Y^0_{17,3}$. This contributes to reduce the complexity of the dominant part by 1 bit. Furthermore, by utilizing the linearity of the F_0 operation, i.e., $Y^0_{17,3} = X^1_{18,3} \oplus X^6_{18,3} \oplus X^7_{18,3}$ we can move more subkey bits from the dominant part. The transformation of the equation $\bigoplus(X^0_{18,4} \oplus SK^0_{69}) = \bigoplus Y^0_{17,3}$ is as follows.

$$\begin{aligned}
&\bigoplus(X^0_{18,4} \oplus SK^0_{69}) = \bigoplus(X^1_{18,3} \oplus X^6_{18,3} \oplus X^7_{18,3}),\\
&\bigoplus(X^0_{18,4} \oplus SK^0_{69}) = \bigoplus(X^1_{19,4} \oplus X^6_{19,4} \oplus X^7_{19,4}\\
&\qquad \oplus Z^1_{18,3} \oplus Z^6_{18,3} \oplus Z^7_{18,3}),\\
&\bigoplus(X^0_{18,4} \oplus SK^0_{69} \oplus X^1_{19,4} \oplus X^6_{19,4} \oplus X^7_{19,4})\\
&\qquad = \bigoplus(Z^1_{18,3} \oplus Z^6_{18,3} \oplus Z^7_{18,3}).
\end{aligned} \tag{3}$$

Finally, we use Eq. (3) for the match to identify the right key candidates. The entire structure is shown in Fig. 9. This reduces the number of subkey bits in the dominant part by 17 bits and the number of ciphertexts by 8 bits, thus the complexity of the attack can be reduced roughly by a factor of 2^{25}.

4.4 Improved Integral Analysis on 22-Round HIGHT

We explain the details of the computation of $\bigoplus(Z^1_{18,3} \oplus Z^6_{18,3} \oplus Z^7_{18,3})$, which is shown in Fig. 10. With the previous method in [17], the complexity is $2^{32} \cdot 2^{56} = 2^{88}$ partial decryptions, while our method, using the bit-wise partial-sum, computes it only with $2^{64.60}$ round function computations.

We first compute $C'_2 \leftarrow F_0(C_2)$ and $WK'_5 \leftarrow F_0(WK_5)$ so that the F_0 function can be excluded from the analysis. Hence, we aim to recover the value of WK'_5 instead of WK_5.

Then, the partial decryption up to $X_{21,3}$ is written as

$$X_{21,3} \leftarrow \Big(C_3 \oplus (SK_{85} \boxplus (C'_2 \oplus WK'_5))\Big).$$

The equation is not exactly the same as 4 equations analyzed in Sect. 3, but the similar procedure can be applied. Namely, for each key guess, the result is computed from the LSB

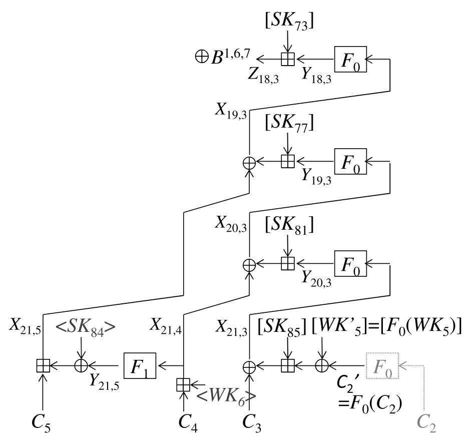


Fig. 10 Computation of $\bigoplus(Z^1_{18,3} \oplus Z^6_{18,3} \oplus Z^7_{18,3})$ in 22-round attack.

to MSB bit by bit and the data is compressed for each bit. Moreover, we use the linear relations in the MSB, thus the data can be compressed with respect to the value of $C^7_3 \oplus C'^7_2$ before the analysis starts and 1-bit guess of $SK^7_{85} \oplus WK'^7_5$ is enough for these two MSBs.

After we obtain the value of $X_{21,3}$ and then the corresponding $Y_{20,3}$, another pattern of the bitwise partial-sum appears for $X_{20,3} = (C_4 \boxplus WK_6) \oplus (Y_{20,3} \boxplus SK_{81})$. However, the analysis is not simple in this time because C_4 is also used to compute $X_{21,5}$ and thus we cannot eliminate the value of C_4 after $X_{20,3}$ is computed. Such a structure makes the attack complicated. In the following, we give the detailed attack procedure to compute $\bigoplus(Z^1_{18,3} \oplus Z^6_{18,3} \oplus Z^7_{18,3})$.

1. Do a precomputation to make 2 look-up tables which return $F_0(x)$ and $F_1(x)$ for a given x.
2. Query 2^{56} plaintexts of the form (A, A, A, A, A, A, A, C) to obtain the corresponding ciphertexts. Count how many times each 4-byte tuple (C_2, C_3, C_4, C_5) appears and pick the ones that appear odd times. As a result, the number of texts to be analyzed is at most $2^{4*8} = 2^{32}$.
3. Convert (C_2, C_3, C_4, C_5) into (C'_2, C_3, C_4, C_5) with the look-up table.
4. Compress the data with respect to $t_1 = C'^7_2 \oplus C^7_3$ and obtain 2^{31} data of $(C'^{0-6}_2, C^{0-6}_3, t_1, C_4, C_5)$. For each 15-bit guess of $(SK^{0-6}_{85}, WK'^{0-6}_5, (SK^7_{85} \oplus WK'^7_5))$, compute $X_{21,3}$ with the bitwise partial-sum technique. The data is compressed to 2^{24} texts of the form $(X_{21,3}, C_4, C_5)$. This is converted into $(Y_{20,3}, C_4, C_5)$ with the look-up table.
5. For each 8-bit guess of WK_6, update 2^{24} data $(Y_{20,3}, C_4, C_5)$ into $(Y_{20,3}, X_{21,4}, C_5)$. Then, compute the corresponding $Y_{21,5}$ and add the value to the data to be analyzed. Hereafter, we regard $X_{21,4}$ and $Y_{21,5}$ are independent. Thus the data size increases to 2^{32}.
6. Compress the data with respect to $t_2 = C^7_5 \oplus Y^7_{21,5}$ and $t_3 = X^7_{21,4} \oplus Y^7_{20,3}$, and obtain 2^{30} data of $(C^{0-6}_5, Y^{0-6}_{21,5}, t_2, X^{0-6}_{21,4}, Y^{0-6}_{20,3}, t_3)$.
7. For each 8-bit guess of SK_{81}, compute $X_{20,3}$ with the bitwise partial-sum technique and compress the data to 2^{23} data of the form $(C^{0-6}_5, Y^{0-6}_{21,5}, t_2, X_{20,3})$. Then, convert the set into $(C^{0-6}_5, Y^{0-6}_{21,5}, t_2, Y_{19,3})$ with the look-up table.
8. Compress the data with respect to $t_4 = t_2 \oplus Y^7_{19,3}$ and obtain 2^{22} data of the form $(C^{0-6}_5, Y^{0-6}_{21,5}, Y^{0-6}_{19,3}, t_4)$.
9. For each 16-bit guess of (SK_{84}, SK_{77}), compute $X_{19,3}$ with the bitwise partial-sum technique without using the optimization of guessing $SK^7_{84} \oplus SK^7_{77}$, and compress the data to 2^8 data of the form $X_{19,3}$. Then, convert the set into $Y_{18,3}$ with the look-up table.
10. For 8-bit guess of SK_{73}, compute $\bigoplus(Z^1_{18,3} \oplus Z^6_{18,3} \oplus Z^7_{18,3})$ in bit-by-bit. Store the result in a table T.

We evaluate the complexity of each step.

- Step 1 requires 2^8 F_0 and F_1 computations, which are negligible.
- Step 2 requires 2^{56} memory access to deal with the ciphertexts. The memory requirement is $2^{32} \cdot 4 = 2^{34}$ bytes.
- Step 3 requires 2^{32} table look-ups.
- In Step 4, because 2 subkeys and 1 modular addition are involved, the complexity of the bitwise partial-sum is 2^{32+n+1}, where $n = 8$. Hence, the complexity is 2^{41} round functions. The data is compressed to 2^{24} and then 2^{24} table look-ups are performed.
- Step 5 requires $2^{15+8+24}$ modular additions and table look-ups, which is less than 2^{47} round functions. The data size increases to 2^{32}.
- Step 6 requires $2^{15+8+32}$ 2-bit XOR operation and table look-ups, which is less than $2^{15+8+32+1} = 2^{56}$ round functions. The data is compressed to 2^{30}.
- In Step 7, the bitwise partial-sum for 1 subkey is applied. Here, the complexity should be analyzed carefully. Because the data compression in the MSB has already finished in Step 6, the optimization of the single-key case discussed in Sect. 3 cannot be applied further. This makes the complexity of Step 7 $8 \cdot 2^{15+8+30+2} = 2^{58}$ round functions. The data is compressed to 2^{23}.
- Step 8 requires $2^{15+8+8+23}$ 1-bit XOR computations, which is less than 2^{54} round functions. The data is compressed to 2^{22}.
- In Step 9, the bitwise partial-sum for 2 subkeys is applied. Here, two optimizations discussed in Sect. 3 cannot be applied:
  1) We guess MSBs of SK_{84} and SK_{77} separately, rather than guessing XOR of them. This is because SK_{84} is the subkey bytes guessed in both of the left-hand side and the right-hand side of Eq. (3), and we later compare all bits of SK_{84} to find the valid matches between two sides of Eq. (3).
  2) Similarly to Step 7, the data has already been compressed with respect to the MSB.
  Thus, the optimization for 2 bits cannot be applied. As a result, the complexity of Step 9 is $2^{15+8+8+22+(n+4)} = 2^{65}$ round functions for $n = 8$. The data is compressed to 2^8.

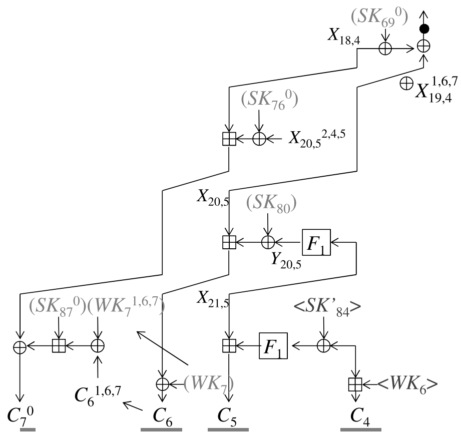


Fig. 11 Computation of left-hand side of Eq. (3) in 22-round attack.

- In Step 10, because 1 subkey and 1 modular addition are involved, the complexity is $2^{8+15+8+8+16+8+1} = 2^{64}$ round functions.

The complexities for steps 1 to 4 are negligible but for 2^{56} memory access in Step 2. The complexity for the remaining part is $2^{47} + 2^{56} + 2^{58} + 2^{54} + 2^{65} + 2^{64} \approx 2^{65.59}$ round functions. After the analysis, we obtain 2^{55} values in T. Note that besides the 1-bit value of $\bigoplus(Z^1_{18,3} \oplus Z^6_{18,3} \oplus Z^7_{18,3})$, we also store 16-bit values of WK_6, SK_{84}, which are later used for the match. Hence, the memory requirement to construct T is $2^{55} * 16$ bits, which is 2^{56} bytes.

4.4.1 Left-Hand Side of Eq. (3)

We also evaluate the complexity to compute the left-hand side of Eq. (3). The entire computation structure is shown in Fig. 11.

This computation only involves 25 bits of the ciphertext and 35 bits of subkeys. Therefore the complexity is at most $2^{25+35} = 2^{60}$ partial decryptions even with the naive method. Because it is obvious that this part can be much lower complexity than the computation of $\bigoplus(Z^1_{18,3} \oplus Z^6_{18,3} \oplus Z^7_{18,3})$, we omit the detailed explanation. However, to demonstrate the impact of the bitwise partial-sum, we show several approaches to optimize the computation.

- Both of C_6^7 and WK_7^7 linearly impact to the final sum exactly twice. Hence, their impacts are canceled. This reduces the data to be analyzed by 1 bit, and the number of key guesses by 1 bit.
- $SK^0_{87}, SK^0_{76}, SK^0_{69}$, and SK^7_{80} linearly impact to the final sum. Hence, 1-bit guess for their XOR value is enough. This reduces the number of key guesses by 3 bits.

In the end, the left-hand side of Eq. (3) is computed with a negligible cost compared to the other part.

4.4.2 Remaining Part

At the very beginning, the partial decryption for the last 5 rounds in Fig. 9 involved 75 subkey bits. We then reduced the number of guessed subkey bits by 1 bit for the left-hand side of Eq. (3) and 3 bits for the right-hand side of Eq. (3) by guessing linear combination of several subkey bits. Hence, the subkey space to be guessed is 71(= 75 − 1 − 4) bits.

The computation of the right-hand side of Eq. (3) involves 55 subkey bits and the computation of the left-hand side of Eq. (3) involves 32 subkey bits. We can check the match of 1 bit of the state with Eq. (3). Moreover, WK_6 and SK_{84} are guessed in both sides of Eq. (3) independently, and thus we can check the match of those 16 bits. In the end, we check the match of 17(= 1 + 16) bits. After the match, we have $55+32-17 = 70$ bits of the candidates for the guessed 71-bit subkey space. In other words, the original 71-bit subkey space is reduced by 1 bit with a single set of 2^{56} plaintexts. Therefore, by iterating the analysis 71 times, the right key for these 71 subkey bits is obtained. After 71 subkey bits are recovered, the remaining key bits can be recovered by the exhaustive search with a cost about $2^{128-71} = 2^{57}$.

To sum up, the data complexity is $71 * 2^{56} \approx 2^{62.15}$ chosen plaintexts. The computational complexity is $71 * 2^{65.59} \approx 2^{71.74}$ round functions, $2^{62.15}$ memory access to deal with the ciphertexts, and 2^{57} 22-round HIGHT computations for the exhaustive search of remaining subkey space, which is about $(2^{71.74}/22 + 2^{57}) \approx 2^{67.28}$ 22-round HIGHT computations. The memory requirement of the attack is 2^{56} bytes.

Note that a tradeoff is available between data and time complexities. With $\mathcal{N}$ sets of 2^{56} plaintexts, 71-bit subkey space is reduced to $2^{71-\mathcal{N}}$ bits. Then, the cost of the exhaustive search becomes $2^{71-\mathcal{N}+57} = 2^{128-\mathcal{N}}$. Hence, the attack is faster than the brute force attack on 128 bits even with a single set of 2^{56} plaintexts.

4.5 New Integral Analysis on 26-Round HIGHT

In [22], Zhang et al. presented another 17-round integral distinguisher. For a set of 2^{56} plaintexts with the form of (A, A, A, C, A, A, A, A), the state after 17 rounds, $(X_{17,7}\|\cdots\|X_{17,0})$, has the form of $(B^0, ?, ?, ?, ?, ?, ?, ?)$. Different from the previous 22-round attack [17], we use this distinguisher. This is because subkey relations in this distinguisher are more advantageous than the other one when 9 rounds are appended to the distinguisher. In fact, we could not attack 26 rounds with the other distinguisher.

With the same transformation as for obtaining Eq. (3), $\bigoplus X^0_{17,7} = 0$ can be transformed as

$$\bigoplus(X^0_{18,0} \oplus SK^0_{71} \oplus X^1_{19,0} \oplus X^6_{19,0} \oplus X^7_{19,0}) = \bigoplus(Z^1_{18,7} \oplus Z^6_{18,7} \oplus Z^7_{18,7}). \tag{4}$$

The computation for the right-hand side of Eq. (4) requires much more complexity than the left-hand side. The partial decryption for obtaining $\bigoplus(Z^1_{18,7} \oplus Z^6_{18,7} \oplus Z^7_{18,7})$ is shown in Fig. 12. The computation involves 96-bit keys $K_0, \ldots, K_3, K_5, \ldots, K_8, K_{10}, K_{11}, K_{14}, K_{15}$ and all ciphertext bytes are related.

We first describe a relatively simple procedure with the

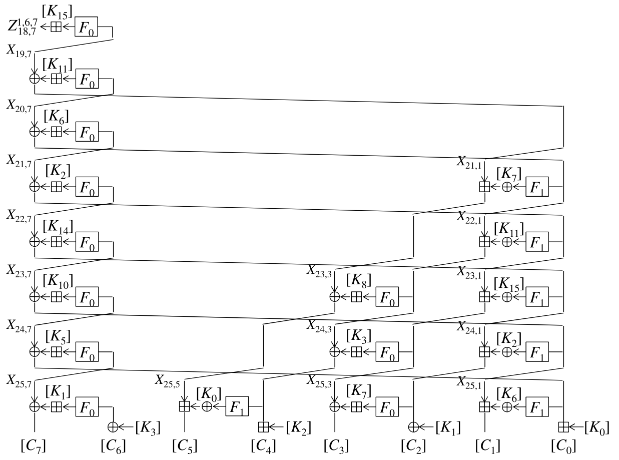


Fig. 12 Partial decryption for $\bigoplus(Z^1_{18,7} \oplus Z^6_{18,7} \oplus Z^7_{18,7})$ in 26-round attack.

bytewise partial-sum. Then, we apply the bitwise partial-sum technique to the dominant part.

1. The analysis stars from at most 2^{64} ciphertexts of $(C_0, \ldots, C_7)$.
2. Guess (K_1, K_3), and compress the data into 2^{56} texts of $(Y_{24,7}, C_5, \ldots, C_0)$.
3. Guess (K_0, K_5, K_6), and compress the data into 2^{48} texts of $(Y_{23,7}, C_5, \ldots, C_2, X_{25,1})$.
4. Guess (K_2, K_7, K_{10}), and compress the data into 2^{40} texts of $(Y_{22,7}, C_5, X_{25,4}, X_{25,3}, X_{24,1})$.
5. Guess (K_{14}, K_{15}), and compress the data into 2^{32} texts of $(Z_{20,7}, X_{24,4}, X_{24,3}, X_{23,1})$.
6. Guess (K_8, K_{11}), and compress the data into 2^{24} texts of $(Z_{20,7}, X_{22,2}, X_{22,1})$.
7. Further compress the data into 2^{16} texts of $(Y_{19,7}, X_{21,1})$.
8. Further compress the data into 2^8 texts of $(Y_{18,7})$.
9. Finally compute $\bigoplus(Z^1_{18,7} \oplus Z^6_{18,7} \oplus Z^7_{18,7})$, and store the result in a table T.

The complexity for each step is as follows.

1. The complexity for Step 1 is 2^{64} memory access.
2. The complexity for Step 2 is less than $2^{16} \cdot 2^{64} = 2^{80}$ round functions.
3. The complexity for Step 3 is less than $2^{16} \cdot 2^{24} \cdot 2^{56} = 2^{96}$ round functions.
4. The complexity for Step 4 is less than $2^{40} \cdot 2^{24} \cdot 2^{48} = 2^{112}$ round functions.
5. The complexity for Step 5 is less than $2^{64} \cdot 2^{16} \cdot 2^{40} = 2^{120}$ round functions.
6. The complexity for Step 6 is less than $2^{80} \cdot 2^{16} \cdot 2^{32} = 2^{128}$ round functions.
7. The complexity for Step 7 is less than $2^{96} \cdot 2^0 \cdot 2^{24} = 2^{120}$ round functions.
8. The complexity for Step 8 is less than $2^{96} \cdot 2^0 \cdot 2^{16} = 2^{112}$ round functions.
9. The complexity for Step 9 is less than $2^{96} \cdot 2^0 \cdot 2^8 = 2^{104}$ round functions.

The entire complexity is the sum of the complexity for each step. The dominant part is Step 6, which requires about 2^{128} round function computations with the bytewise partial-sum. We explain that the bitwise partial-sum can be applied to Step 6 to reduce the complexity.

Step 6 starts from 2^{32} texts of $(Z_{20,7}, X_{24,4}, X_{24,3}, X_{23,1})$, and the goal is obtaining 2^{24} texts of $(Z_{20,7}, X_{22,2}, X_{22,1})$ with guessing two subkeys (K_8, K_{11}). At first, we update 2^{32} texts into $(Z_{20,7}, X_{24,4}, X_{24,3}, Y_{22,1})$, and we guess 1-byte key K_8 and update 2^{32} texts into $(Z_{20,7}, X_{22,2}, X_{24,3}, Y_{22,1})$. Up to here, the complexity for the guess of K_8 is less than $2^{80} \cdot 2^8 \cdot 2^{32} = 2^{120}$ round functions. We then apply the bitwise partial-sum to guess K_{11}. First of all, by exploiting the MSB, 2^{32} texts is compressed into 2^{31} texts of $(Z_{20,7}, X_{22,2}, X^{0-6}_{24,3}, Y^{0-6}_{22,1}, t^7)$, where t^7 is $X^7_{24,3} \oplus Y^7_{22,1}$. Then, compute $X_{22,1}$ bit-by-bit from the LSB to MSB to obtain 2^{24} texts of $(Z_{20,7}, X_{22,2}, X_{22,1})$. This is a single-key case with 1 modular addition. The complexity is about $n \cdot (2^{88+32+1}) = 2^{124}$ round functions, where $n = 8$.

Finally, for a single set of chosen plaintexts, the sum of the complexity for all steps is 2^{64} memory access and $2^{80}+2^{96}+2^{112}+2^{120}+(2^{120}+2^{124})+2^{120}+2^{112}+2^{104} \approx 2^{124.25}$ round functions, which is $2^{124.25}/26 \approx 2^{119.55}$ 26-round HIGHT computations. The data complexity is 2^{56} chosen plaintexts. For each 2^{96} guess, we store guessed 12 bytes and 1-bit information for the match of the sum. Hence, the memory requirement is about $12 \cdot 2^{96} \approx 2^{99.58}$ bytes.

The computation for the left-hand side of Eq. (4) is depicted in Fig. 13, which involves 89-bit keys $K_0, \ldots, K_4, K_6, K_7, K_8, K^0_9, K_{11}, K_{12}, K_{15}$ and 57 bits of ciphertext are related. The complexity is significantly smaller than the right-hand side of Eq. 4. For each 2^{89} guess, we obtain the guessed 89 key bits and 1-bit information for the match of the sum.

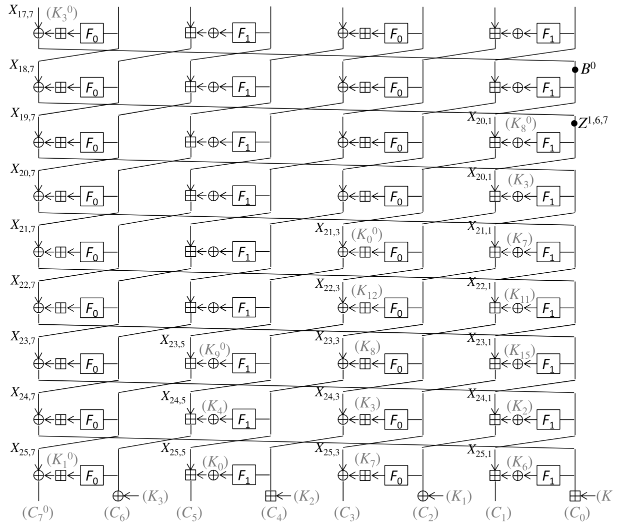


Fig. 13 Computation Structure of Left-hand side of Eq. (4).

4.5.1 Remaining Part

The partial decryption for the last 9 rounds involves 113 subkey bits, which are all bits but for 7 bits of K_9 and 8 bits of K_{13}. The computation of the right-hand side of Eq. (4) involves 96 subkey bits and the computation of the left-hand side of Eq. (4) involves 89 subkey bits. Both sides include 72 subkey bits in common.

With one plaintext set, the right-hand side of Eq. (4) provides 2^{96} candidates and the left-hand side of Eq. (4) provides 2^{89} candidates. Hence, we check the match of 2^{185} pairs. We can check the match of 72 subkey bits and 1 bit of the state with Eq. (4), in total 73 bits. As a result, $2^{185-73} = 2^{112}$ subkey candidates are obtained for the 113-bit subkey space. In other words, the 113-bit subkey space is reduced by 1 bit with a single set of 2^{56} plaintexts. Note that we need to store the suggested 2^{112} subkey candidates, which is $12 * 2^{112} = 2^{115.58}$ bytes. This is the bottle-neck of the memory complexity.

Suppose that we use $\mathcal{N}$ sets of 2^{56} plaintexts. Then, 113-bit subkey space is reduced to $2^{113-\mathcal{N}}$ bits with a time complexity of $\mathcal{N} \cdot 2^{119.55}$ 26-round HIGHT computations. We perform the exhaustive search for the remaining subkey space, which is $2^{113-\mathcal{N}}$ bits plus uninvolved 15 bits. Hence, the cost of the exhaustive search is $2^{113-\mathcal{N}+15} = 2^{128-\mathcal{N}}$ 26-round HIGHT computations. To balance the time complexity of two parts, we obtain the following equation:

$$\mathcal{N} \cdot 2^{119.55} = 2^{128-\mathcal{N}},$$
$$\Rightarrow \mathcal{N} + \log_2 \mathcal{N} = 8.45.$$

$\mathcal{N} = 6$ makes the above equation most balanced. As a result, the data complexity is $6 * 2^{56} \approx 2^{58.58}$ chosen plaintexts. The time complexity for analyzing 113 subkey bits is $\mathcal{N} \cdot 2^{119.55} \approx 2^{122.13}$ 26-round HIGHT computations. The time complexity for the exhaustive search is $2^{128-\mathcal{N}} = 2^{122}$ 26-round HIGHT computations. In total, the final time complexity is $2^{122.13} + 2^{122} \approx 2^{123.07}$ 26-round HIGHT computations. The memory complexity is $2^{115.58}$ bytes for storing the 2^{112} suggested subkey candidates after analyzing the first set of 2^{56} plaintexts.

5. Concluding Remarks

In this paper, we presented the bitwise partial-sum technique that can reduce the complexity of the integral analysis against ARX based designs. The technique computes equations bit-by-bit from the LSB to the MSB and keeps the data size as small as possible by compressing the data with respect to the value of the guessed bit position and the carry value to the next bit position. We applied this technique to the integral analysis on HIGHT. We also proposed an improvement specific to HIGHT, which exploits the linearity of the computations inside the round function. Due to these effort, we could improve the complexity of the previous 22-round attack, and moreover, the number of attacked rounds were extended up to 26 steps. Although our attack only can work for a smaller number of rounds compared to impossible differential attack, we believe that several new observations together with a significant improvement for the integral analysis lead to a deeper understanding.

Acknowledgments

The authors would like to thank one of the anonymous reviewers of IEICE Trans. for carefully checking technical details and pointing out mistakes about the complexity evaluation. Lei Wang is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

[1] J.-P. Aumasson, G. Leurent, W. Meier, F. Mendel, N. Mouha, R.C.-W. Phan, Y. Sasaki, and P. Susil, "Tuple cryptanalysis of ARX with application to BLAKE and Skein," ECRYPT II Hash Workshop, 2011.

[2] A. Biryukov and A. Shamir, "Structural cryptanalysis of SASAS," Birgit Pfitzmann, editor, Advances in Cryptology — Eurocrypt 2001, Lect. Notes Comput. Sci., vol.2045, pp.394–405, Springer-Verlag, Berlin, Heidelberg, New York, 2001.

[3] J. Chen, M. Wang, and B. Preneel, "Impossible differential cryptanalysis of the lightweight block ciphers TEA, XTEA and HIGHT," A. Mitrokotsa and S. Vaudenay, ed., Africacrypt 2012, Lect. Notes Comput. Sci., vol.7374, pp.117–137, Springer-Verlag, Berlin, Heidelberg, New York, 2012.

[4] J. Daemen, L.R. Knudsen, and V. Rijmen, "The block cipher Square," E. Biham, ed., Fast Software Encryption 1997, Lect. Notes Comput. Sci., vol.1267, pp.149–165, Springer-Verlag, Berlin, Heidelberg, New York, 1997.

[5] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved cryptanalysis of Rijndael," B. Schneier, ed., FSE 2000, Lect. Notes Comput. Sci., vol.1978, pp.213–230, Springer-Verlag, Berlin, Heidelberg, New York, 2000.

[6] 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 hash function competition (Round 2), Version 1.2 — 15 Sept. 2009. Available online: http://www.skein-hash.info

[7] 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," L. Goubin and M. Matsui, ed., CHES 2006, Lect. Notes Comput. Sci., vol.4249, pp.46–59, Springer-Verlag, Berlin, Heidelberg, New York, 2006.

[8] ISO/IEC 18033-3:2010, Information technology — Security techniques — Encryption Algorithms — Part 3: Block ciphers, 2010.

[9] L.R. Knudsen and D. Wagner, "Integral cryptanalysis," J. Daemen and V. Rijmen, ed., FSE 2002, Lect. Notes Comput. Sci., vol.2365, pp.112–127, Springer-Verlag, Berlin, Heidelberg, New York, 2002.

[10] B. Koo, D. Hong, and D. Kwon, "Related-key attack on the full HIGHT," K.H. Rhee and D. Nyang, ed., ICISC 2010, Lect. Notes Comput. Sci., vol.6829, pp.49–67, Springer-Verlag, Berlin, Heidelberg, New York, 2011.

[11] Korea Internet and Security Agency, HIGHT Algorithm Specification, 2009.

[12] J. Lu, Cryptanalysis of block ciphers, PhD thesis, Royal Holloway, University of London, England, July 2008.

[13] S. Lucks, "The saturation attack — A bait for Twofish," M. Matsui, ed., Fast Software Encryption 8th International Workshop, FSE 2001, Lect. Notes Comput. Sci., vol.2355, pp.1–15, Springer-Verlag, Berlin, Heidelberg, New York, 2001.

[14] J. Nakahara, Jr., D.S. de Freitas, and R.C.-W. Phan, "New multiset attacks on rijndael with large blocks," E. Dawson and S. Vaudenay, ed., Mycrypt 2005, Lect. Notes Comput. Sci., vol.3715, pp.277–295, Springer-Verlag, Berlin, Heidelberg, New York, 2005.

[15] R.M. Needham and D.J. Wheeler, "TEA extensions," Technical report, Computer Laboratory, University of Cambridge, Oct. 1997.

[16] O. Özen, K. Varici, C. Tezcan, and Ç. Kocair, "Lightweight block ciphers revisited: Cryptanalysis of reduced round PRESENT and HIGHT," C. Boyd and J.M. González Nieto, ed., ACISP 2009, Lect. Notes Comput. Sci., vol.5594, pp.90–107, Springer-Verlag, Berlin, Heidelberg, New York, 2009.

[17] Y. Sasaki and L. Wang, "Meet-in-the-middle technique for integral attacks against feistel ciphers," L.R. Knudsen and H. Wu, ed., Selected Areas in Cryptography SAC 2012, Lect. Notes Comput. Sci., vol.7707, pp.234–251, Springer-Verlag, Berlin, Heidelberg, New York, 2012.

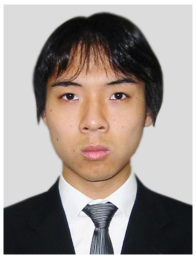
[18] T. Suzaki, K. Minematsu, S. Morioka, and E. Kobayashi, "TWINE: A lightweight block cipher for multiple platforms," L.R. Knudsen and H. Wu, ed., Selected Areas in Cryptography SAC 2012, Lect. Notes Comput. Sci., vol.7707, pp.339–354, Springer-Verlag, Berlin, Heidelberg, New York, 2012.

[19] L. Wen, M. Wang, A. Bogdanov, and H. Chen, "Multidimensional zero-correlation attacks on lightweight block cipher HIGHT: Improved cryptanalysis of an ISO standard," Inf. Process. Lett., vol.114, no.6, pp.322–330, June 2014. Available online 21 Jan. 2014.

[20] W. Wu and L. Zhang, "LBlock: A lightweight block cipher," J. Lopez and G. Tsudik, ed., ACNS 2011, Lect. Notes Comput. Sci., vol.6715, pp.327–344, Springer-Verlag, Berlin, Heidelberg, New York, 2011.

[21] M.R. Z'aba, H. Raddum, M. Henricksen, and E. Dawson, "Bit-pattern based integral attack," K. Nyberg, ed., FSE 2008, Lect. Notes Comput. Sci., vol.5086, pp.363–381, Springer-Verlag, Berlin, Heidelberg, New York, 2008.

[22] P. Zhang, B. Sun, and C. Li, "Saturation attack on the block cipher HIGHT," J.A. Garay, A. Miyaji, and A. Otsuka, ed., CANS, Lect. Notes Comput. Sci., vol.5888, pp.76–86, Springer-Verlag, Berlin, Heidelberg, New York, 2009.



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, SECRYPT 2012, and IWSEC 2012.

Lei Wang received the M.E. and Ph.D. from The University of Electro-Communications in 2009, and 2011. 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.