Cryptanalysis of BAKSHEESH Block Cipher

Shengyuan Xu¹, Siwei Chen $(\boxtimes)^{2,5}$, Xiutao Feng³, Zejun Xiang^{2,5}, and Xiangyong Zeng^{4,5}

 ¹ Department of Fundamental Courses, Shandong University of Science and Technology, Taian, Shandong, China xushengyuan@sdust.edu.cn
 ² School of Cyber Science and Technology, Hubei University, Wuhan, Hubei, China chensiwei_hubu@163.com, xiangzejun@hubu.edu.cn

³ Key Laboratory of Mathematics Mechanization, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing, China fengxt@amss.ac.cn

⁴ Faculty of Mathematics and Statistics, Hubei Key Laboratory of Applied

Mathematics, Hubei University, Wuhan, Hubei, China xzeng@hubu.edu.cn

⁵ Key Laboratory of Intelligent Sensing System and Security, Ministry of Education, Hubei University, Wuhan, Hubei, China

Abstract. BAKSHEESH is a lightweight block cipher following up the wellknown cipher GIFT-128, which uses a 4-bit SBox that has a non-trivial Linear Structure (LS). Also, the Sbox requires a low number of AND gates that makes BAKSHEESH stronger to resist the side channel attacks compared to GIFT-128. In this paper, we give the first third-party security analysis of BAKSHEESH from the traditional attacks perspective: integral, differential and linear attacks. Firstly, we propose a framework for integral attacks based on the properties of BAKSHEESH's Sbox and its inverse. By this, we achieve the 9- and 10-round practical key-recovery attacks, and give a 15-round theoretical attack. Secondly, we re-evaluate the security bound against differential cryptanalysis, correcting two errors from the original paper and presenting a key-recovery attack for 19 rounds. At last, for linear cryptanalysis, we develop an automated model for key-recovery attacks and then demonstrate a key-recovery attack for 21 rounds. We stress that our attacks cannot threaten the full-round BAKSHEESH, but give a deep understanding on its security.

Keywords: BAKSHEESH \cdot Lightweight Block Cipher \cdot Security Evaluation \cdot Key-recovery Attacks

1 Introduction

In today's world, the Internet of Things (IoT) is pivotal across various domains, including agriculture, healthcare, industry, and transportation. Despite their widespread use, IoT devices often suffer from resource constraints limited computing power and energy which renders traditional cryptographic methods insufficient for safeguarding data. To overcome these challenges, the field of lightweight cryptography has emerged, resulting in the creation of a range of efficient block ciphers. Notable examples include PRESENT [8], CLEFIA [16], PRINCE [10], SIMON [3], SPECK [3], SKINNY [4], and GIFT [2], each drawing significant interest from researchers.

Recently, a lightweight block cipher called BAKSHEESH [1] was proposed by Baksi et al., building upon the well-known cipher GIFT-128. BAKSHEESH runs for 35 rounds, and the block and key are both 128 bits. Its round function comprises four distinct steps: SubCells: 32 4-bit SBoxes are applied to the state; PermBits: permutes the bits of the state; AddRoundConstants: involving the XOR operation with a 6-bit constant and an additional bit; and AddRoundKey: the round key is XORed with the state. BAKSHEESH's Sbox has a very low number of AND gates so that it is suitable to side channel counter measures (when compared to GIFT-128) and other niche applications. But the LS in its Sbox seems more aggressive than GIFT-128 from the security against traditional attacks like differential cryptanalysis perspective. Aside from the basic security analysis provided by the designers themselves, there is a notable lack of concrete third-party security evaluations in the existing literature. To address this gap, this paper presents intergral cryptanalysis, differential cryptanalysis and linear cryptanalysis of the BAKSHEESH cipher, offering a comprehensive examination of its security.

Differential cryptanalysis [7] and linear cryptanalysis [15] are two of the most important cryptanalysis in block ciphers. At CRYPTO '90, Biham and Shamir introduced the concept of differential cryptanalysis, achieving the first theoretical attack on the full-round Data Encryption Standard (DES). Subsequently, Matsui proposed linear cryptanalysis at EUROCRYPT '93, enhancing the fullround attack on DES. Since these seminal contributions, several useful techniques have been developed based on them, such as related-key differential attack [5], truncated differential attack [12], impossible differential attack [6] and zero correlation attack [9].

The integral attack [13] is another valuable method for assessing the security of symmetric-key primitives. It was introduced by Daemen et al. in 2002 to evaluate the security of the block cipher Square [11] and was later formalized by Knudsen and Wagner. The integral attack involves two main steps: the construction of an integral distinguisher followed by a key-recovery process. In 2015, a novel technique for constructing integral characteristics was introduced [19] which presented a new concept known as the division property by generalizing the integral property originally discussed in [13]. By modeling the propagation of the division property as a Mixed Integer Linear Programming (MILP) problem, Xiang et al. [20] developed an automated tool to facilitate the search for integral distinguishers.

1.1 Our Contributions

In this paper, we present a comprehensive third-party cryptanalysis and propose several key-recovery attacks on BAKSHEESH. Given that integral attacks yield the longest practical key-recovery, we utilize integral cryptanalysis for our evaluation and obtain practical key-recovery attacks. Morever, we also search differential/linear trails automated and provide theoretical key-recovery attacks by leveraging distinguishers based on differential trails and linear trails. Our results are summarized in Table 1, and the detailed contributions are outlined as follows.

Practical integral attacks. Based on the properties of BAKSHEESH's Sbox and its inverse, we propose a framework for (r + 2)-round key-recovery attacks of BAKSHEESH using *r*-round integral distinguishers. Moreover, we introduce an optimized search strategy based on the MILP-aided division property to identify integral distinguishers more efficiently, which are suitable for the attack framework. As a result, we find a 7-round and an 8-round distinguishers with data complexity 2^{14} and 2^{30} , and utilize them to achieve the 9-round and 10-round practical attacks, repsectively, which can be verified by experiments. Also, we present a 15-round theoretical attack with time complexity 2^{127} based on a 13round distinguisher.

SAT-aided re-evaluation on the security bound against differential/linear cryptanalysis. We utilized automated tools to reassess the optimal differential bounds for BAKSHEESH and identified two errors in the original work concerning differential trails. Additionally, we present a key-recovery attack for 19-round BAKSHEESH based on the differential distinguisher we found. Furthermore, we conducted a thorough automated search for full-round linear trails and verified the correctness of the original results. Finally, by characterizing the key-recovery state within the SAT models, we provide a key-recovery attack for 21-round BAKSHEESH.

Method	#R	Time	Data	Memory	Ref.
Integral	9	$2^{32.01}$	$2^{15.59}$	-	Sect. 3.3
Integral	10	$2^{32.99}$	$2^{31.59}$	$2^{24.59}$	Sect. 3.3
Integral	15	2^{127}	2^{127}	2^{23}	Sect. 3.3
Differential	19	2^{121}	2^{122}	2^{64}	Sect. 4.2
Linear	21	$2^{127.67}$	2^{120}	2^{12}	Sect. 4.4

Table 1. Summary on our attacks for BAKSHEESH. #R: number of attack round.

1.2 Organization of This Paper

This paper is organized as follows: Firstly some necessary notations and preliminaries are introduced in Section 2. In Section 3, integral distinguishers are constructed using automated tools, and practical key-recovery attacks are proposed. Section 4 re-evaluates the security bounds against differential and linear cryptanalysis using the SAT method. Finally, we conclude this paper in Section 5. The source codes are publicly available at https://github.com/damash/ Cryptanalysis-of-BAKSHEESH-Block-Cipher. 4 Authors Suppressed Due to Excessive Length

2 Preliminaries

In this section, we start by providing a brief overview of some notations in Table 2, then give the specification on BAKSHEESH block cipher as well as some basis on integral, differential, and linear cryptanalyses.

Table 2. The notations used throughout the paper. Moreover, X^{r+1} is also the *r*-round output state for $1 \le r \le 34$.

Notation	Description
\mathbb{F}_2	The finite field only contains two elements, i.e., $\{0, 1\}$
\mathbb{F}_2^n	An <i>n</i> -dimensional vectorial space defined over \mathbb{F}_2
\land,\lor,\oplus	Bit-wise AND, OR, XOR
K	The master key
$RK^r = (rk_{127}^r, \cdots, rk_0^r)$	The <i>r</i> -round subkey for $1 \le r \le 35$
$X = (x_{127}, \cdots, x_0)$	The 128-bit internal state
$X^0 = (x_{127}^0, \cdots, x_0^0)$	The 128-bit plaintext
$X^r = (x_{127}^r, \cdots, x_0^r)$	The input of r-round SubCells for $1 \le r \le 35$
$Y^r = (y_{127}^r, \cdots, y_0^r)$	The output of r-round SubCells for $1 \le r \le 35$
S_i	The i -th Sbox

2.1 BAKSHEESH Block Cipher

BAKSHEESH, proposed by Baksi *et al.* [1] at 2023, is a lightweight block cipher that designed by following up the popular block cipher GIFT-128 [2]. The block and key of BAKSHEESH are both 128 bits, and the full iteration round is 35. The round function of BAKSHEESH is based on SPN structure, and consists of the following 4 steps:

- SubCells: applying 32 parallel 4×4 Sboxes to the internal state, namely, the state is updated as:

 $X \leftarrow S(x_{127}||x_{126}||x_{125}||x_{124})|| \cdots ||S(x_7||x_6||x_5||x_4)||S(x_3||x_2||x_1||x_0).$

Table 3 gives the truth table of the Sbox.

Table 3. The truth table of BAKSHEESH's Sbox.

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

- **PermBits**: permute the bit positions of internal state as $x_{P(i)} \leftarrow x_i$ for $0 \le i \le 127$, where P is the permutation as given in Table 4.

0	33	66	99	96	1	34	67	64	97	2	35	32	65	98	3
4	37	70	103	100	5	38	71	68	101	6	39	36	69	102	7
8	41	74	107	104	9	42	75	72	105	10	43	40	73	106	11
12	45	78	111	108	13	46	79	76	109	14	47	44	77	110	15
16	49	82	115	112	17	50	83	80	113	18	51	48	81	114	19
20	53	86	119	116	21	54	87	84	117	22	55	52	85	118	23
24	57	90	123	120	25	58	91	88	121	26	59	56	89	122	27
28	61	94	127	124	29	62	95	92	125	30	63	60	93	126	31

Table 4. The bit permutation P of BAKSHEESH. For instance, P(1) = 33, P(16) = 4.

- AddRoundConstants: XORing a 6-bit constant $C^r = (c_{106}^r, c_{67}^r, c_{35}^r, c_{19}^r, c_{13}^r, c_8^r)$ for $1 \le r \le 35$ to the 6 bits $x_8, x_{13}, x_{19}, x_{35}, x_{67}, x_{106}$, i.e.,

$$x_8 \leftarrow x_8 \oplus c_8^r, \quad x_{13} \leftarrow x_{13} \oplus c_{13}^r, \quad x_{19} \leftarrow x_{19} \oplus c_{19}^r, x_{35} \leftarrow x_{35} \oplus c_{35}^r, \quad x_{67} \leftarrow x_{67} \oplus c_{67}^r, \quad x_{106} \leftarrow x_{106} \oplus c_{106}^r$$

where the round constants C^r are given respectively by: (2, 33, 16, 9, 36, 19, 40, 53, 26, 13, 38, 51, 56, 61, 62, 31, 14, 7, 34, 49, 24, 45, 54, 59, 28, 47, 22, 43, 20, 11, 4, 3, 32, 17, 8). For example, $C^1 = 2$, $(c_{106}^1, c_{67}^1, c_{15}^1, c_{19}^1, c_{13}^1, c_8^1) = (0, 0, 0, 0, 1, 0).$

- AddRoundKey: XORing a 128-bit round key $RK^r = (rk_{127}^r, rk_{126}^r, \cdots, rk_0^r)$ for $0 \le r \le 35$ to the state, i.e., $X^{r+1} \leftarrow X^r \oplus RK^r$. Note that RK^0 is the whitening key.

For the key schedule, BAKSHEESH uses the master key as the whitening key, and the round keys are simply generated by circular right rotation on the master key. In particular, the r-th round key RK^r is represented as

$$RK^r = K \gg r$$
, for $0 \le r \le 35$.

In addition, we give two properties about BAKSHEESH's Sbox that will be used in our attacks.

Property 1. For BAKSHEESH's Sbox, there is a linear trail $1000 \rightarrow 0111$ with correlation 1.

Property 2. For the inverse of BAKSHEESH's Sbox, there is a differential transition $1111 \rightarrow 1000$ with probability 1.

2.2 Integral Cryptanalysis

Integral cryptanalysis, inspired by square attack [11], was originally proposed by Knudsen and Wagner [13] at FSE '02. The core idea is to find a set of plaintexts such that the state after several rounds have a certain integral property, i.e., the XOR sum of the all corresponding states at some or all bit positions is equal to 0. Specifically, the set of plaintexts is generated by traversing all the values of

6 Authors Suppressed Due to Excessive Length

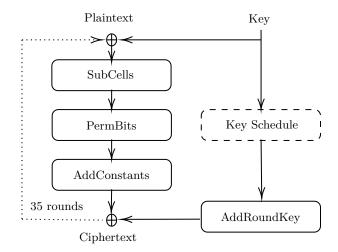


Fig. 1. Schematic of BAKSHEESH

some input bits called *active* bits and fixing the other input bits as constants. By XORing all the corresponding outputs, if some bit positions of the XOR sum are key-independently equal to zero, then we say these output bits are *balanced* bits or they have the balanced integral property. Also, the other bit positions of the XOR sum, whose values are always inconstant with different keys, are called *unknown* bits.

For a block cipher E with the block size of n, \mathbb{D} is a set of plaintexts generated by d (d < n) active bits. If the *r*-round output is $CT = E^r(PT, K)$ and has a balanced integral property at bit position i, then we can use the XOR sum of CT_i under \mathbb{D} to get

$$\bigoplus_{PT\in\mathbb{D}} CT_i = 0,$$

which can be utilized to distinguish the *r*-round E from a pseudorandom permutation. In general, the most significant step of integral cryptanalysis is to find a good integral distinguisher. Previously, a common way to construct an integral distinguisher was to investigate the structure property of block ciphers. Since the division property introduced [19], searching integral distinguishers for block ciphers has been automated with the help of MILP or SAT/SMT based tools [17, 20].

2.3 Differential Cryptanalysis

Differential cryptanalysis is a method of key-recovery that utilizes differential trails with high-probability to construct differential distinguishers. This method was introduced by Biham and Shamir [7]. The core idea is to distinguish an encryption algorithm from a random permutation by studying the effect of plain-

text pair differences on ciphertext pair differences. The definition and probability calculation of a differential trail are as follows:

Definition 1. An r-round differential trail Ω is defined as the following sequence of differences:

 $\alpha_0 \to \alpha_1 \to \cdots \to \alpha_r,$

where α_0 is the difference of the plaintext pair X and X', i.e., $\alpha_0 = X \oplus X'$ and $\alpha_i, 1 \leq i \leq r$, is the difference of the output ciphertext pair after the *i*-th round.

The core of differential cryptanalysis is finding high-probability *r*-round differential trails. To allow the calculation of high-round differential trail probabilities through a round-by-round approach, Lai et al. proposed the "Random Equivalence Hypothesis" [14].

Definition 2. For most keys, the differential propagation probability under a fixed key is approximately equivalent to the differential propagation probability under the assumption that each round key is independent and uniformly distributed.

Therefore, for a given differential trail, under the assumption that the round keys are uniformly randomly distributed, the approximate probability can be calculated as the product of the differential probabilities for each round:

Definition 3. The probability P^{Ω} of an r-round differential trail Ω can be calculated as follows:

$$P^{\Omega} = P_1^{\Omega} \times P_2^{\Omega} \times \dots \times P_r^{\Omega},$$

where P_i^{Ω} is the propagation probability of the differential from the input to the output of the *i*-th round.

2.4 Linear Cryptanalysis

Linear cryptanalysis is a method that exploits linear approximations to construct linear distinguishers and recover the key [15]. The linear approximation of a cipher can be represented as:

$$\Gamma_P \cdot P \oplus \Gamma_C \cdot C = \Gamma_K \cdot K,\tag{1}$$

where Γ_P , Γ_C and Γ_K are called the linear masks of P, C and K, respectively. Denote $\Gamma_K \cdot K = \kappa$, if Eq. (1) holds with probability $1/2 \pm \epsilon$, Matsui showed that with about $|\epsilon|^{-2}$ plaintexts, the one bit information κ of the key can be recovered.

Let f be a transformation over \mathbb{F}_2^n . A linear approximation over f consists of a pair (α, β) of selection vectors over \mathbb{F}_2^n , known as the input mask and output mask, respectively. The correlation $C(\alpha, \beta)$ of a linear approximation (α, β) is the correlation between the Boolean functions $a^T \cdot x$ and $b^T \cdot f(x)$, defined as:

$$C(\alpha,\beta) = \frac{\left|\left\{x \in \mathbb{F}_2^n \left| \alpha^T x + \beta^T f(x) = 0\right\}\right|}{2^{n-1}} - 1.$$

Authors Suppressed Due to Excessive Length

8

Similar to a differential trail, an r-round linear trail is defined as a sequence of linear masks. A linear trail can be represented as

$$\alpha_0 \to \alpha_1 \to \cdots \to \alpha_r,$$

where α_0 is the input mask and α_r is the output mask. This can be extended to more than two random binary variables, X_1 to X_n , with probabilities $p_1 = 1/2 + \varepsilon_1$ to $p_n = 1/2 + \varepsilon_n$. Matsui's piling-up lemma [15] can be used to calculate the probability bias for r rounds.

Lemma 1. (Matsui's piling-up lemma [15]) For random binary variables X_1, X_2, \ldots, X_n and $X_1 \oplus X_2 \oplus \cdots \oplus X_n = 0$, the total bias can be derived as

$$\varepsilon = 2^{n-1} \prod_{i=1}^{n} \varepsilon_i,$$

where ε_i represents the bias of X_i .

3 Integral Attacks on Round-reduced BAKSHEESH

In this section, we will first give an overview of our framework for mounting an (r + 2)-round key-recovery attack using a given *r*-round integral distinguisher. Based on this, we will introduce how to find proper distinguishers, and then use the found 7- and 8-round distinguishers to achieve 9- and 10-round practical attacks, and a 13-round distinguisher to mount a theoretical 15-round key-recovery attack.

3.1 An Overview of the (r+2)-round Key-recovery Attack

In our attack, the main idea is to exploit the so-called *equivalent* integral property to extend an *r*-round integral distinguisher by one round and then recover the (r + 2)-round subkey. The equivalent integral property of BAKSHEESH is illustrated as following proposition.

Proposition 1. For any $0 \le i \le 31$ and $1 \le r \le 34$, the integral property of the bit x_{4i+3}^r is equivalent to that of $x_{P(4i)}^{r+1} \oplus x_{P(4i+1)}^{r+1} \oplus x_{P(4i+2)}^{r+1}$, i.e., given an arbitrary set of plaintexts denoted by \mathbb{D} , we have

$$\bigoplus_{X^0 \in \mathbb{D}} x^r_{4i+3} = \bigoplus_{X^0 \in \mathbb{D}} (x^{r+1}_{P(4i)} \oplus x^{r+1}_{P(4i+1)} \oplus x^{r+1}_{P(4i+2)}).$$
(2)

Proof. From Property 1, we know $x_{4i+3}^r = y_{4i+2}^r \oplus y_{4i+1}^r \oplus y_{4i}^r$ always holds in the *i*-th Sbox S_i of the *r*-th round. Moreover, the round function indicates that

$$x_{P(j)}^{r+1} = y_{P(j)}^r \oplus rk_{P(j)}^{r+1} \oplus rc_{P(j)}^r, 0 \le j \le 127,$$
(3)

where $rc_{P(j)}^r = c_{P(j)}^r$ if $P(j) \in \{8, 13, 19, 35, 64, 106\}$, otherwise $rc_{P(j)}^r = 0$. Therefore, the bit x_{4i+3}^r can be represented as

$$x_{4i+3}^r = x_{P(4i)}^{r+1} \oplus x_{P(4i+1)}^{r+1} \oplus x_{P(4i+2)}^{r+1} \oplus ck_i^r,$$
(4)

where $ck_i^r = rk_{P(4i)}^{r+1} \oplus rk_{P(4i+1)}^{r+1} \oplus rk_{P(4i+2)}^{r+1} \oplus rc_{P(4i)}^{r+1} \oplus rc_{P(4i+1)}^{r+1} \oplus rc_{P(4i+2)}^{r+1}$. Note that ck_i^r is a constant determined by the subkey and round constant of (r+1)-th round, which gives that

$$\bigoplus_{X^0 \in \mathbb{D}} ck_i^r = 0.$$

Thus, we have

$$\bigoplus_{X^{0} \in \mathbb{D}} x_{4i+3}^{r} = \bigoplus_{X^{0} \in \mathbb{D}} (x_{P(4i)}^{r+1} \oplus x_{P(4i+1)}^{r+1} \oplus x_{P(4i+2)}^{r+1} \oplus ck_{i}^{r}) \\
= \bigoplus_{X^{0} \in \mathbb{D}} (x_{P(4i)}^{r+1} \oplus x_{P(4i+1)}^{r+1} \oplus x_{P(4i+2)}^{r+1}).$$
(5)

Explore the related key bits. For a given r-round distinguisher, assume that its output bit at the position 4i+3 ($0 \le i \le 31$), i.e., x_{4i+3}^{r+1} has the balanced integral property, then we can filter the guessed values of partial bits of the subkey RK^{r+2} that related to $x_{P(4i)}^{r+1}$, $x_{P(4i+1)}^{r+1}$ and $x_{P(4i+2)}^{r+1}$ by detecting the balanced property of $x_{P(4i)}^{r+2} \oplus x_{P(4i+1)}^{r+2} \oplus x_{P(4i+2)}^{r+2}$, whose value can be computed by the known (r+2)th round output, i.e., X^{r+3} and the guessed rk^{r+2} . Specifically, for the instance of $x_{P(4i+j)}^{r+2}$ ($j \in \{0,1,2\}$), we first need to compute the 4 bits $(y_{4\ell_j+3}^{r+2}, y_{4\ell_j+1}^{r+2}, y_{4\ell_j+1}^{r+2}, y_{4\ell_j+1}^{r+2})$ as

$$y_{4\ell_j+m}^{r+2} = x_{P(4\ell_j+m)}^{r+3} \oplus rk_{P(4\ell_j+m)}^{r+2}, 0 \le m \le 3$$
(6)

according to the round function, where $\ell_j^i = \left\lfloor \frac{P(4i+j)}{4} \right\rfloor$ and $rk_{P(4\ell_j^i+m)}^{r+2}$ is a guessed key bit. Then by inverting the Sbox $S_{\ell_j^i}$, the value of $x_{P(4i+j)}^{r+2}$ is obtained. It is worth noting that we omit the AddRoundConstants step here for the sake of simplification, but it does not matter as we can combine this step with AddRoundKey. Therefore, there are in total 12 key bits that need to be guessed for detecting the balanced property of x_{4i+3}^{r+1} . We list the key bits of RK^{r+2} corresponding to each $x_{4i+3}^{r+1}(0 \le i \le 31)$ in Table 5.

Denote \mathbb{K}_i $(0 \leq i \leq 31)$ the set that contains the bit positions of RK^{r+2} corresponding to x_{4i+3}^{r+1} , we have an observation from Table 5 as follows.

Observation 1. The union of any b ($b \ge 2$) sets among the s-th ($0 \le s \le 7$) set tuple ($\mathbb{K}_{4s}, \mathbb{K}_{4s+1}, \mathbb{K}_{4s+2}\mathbb{K}_{4s+3}$) is a constant set denoted by \mathbb{K}'_s that contains 16 bit positions.

X^{r+1}	Guessed key RK^{r+2}	Set	Constant Set						
3	0, 33, 66, 99, 8, 41, 74, 107, 16, 49, 82, 115	\mathbb{K}_0							
7	24, 57, 90, 123, 0, 33, 66, 99, 8, 41, 74, 107	\mathbb{K}_1	\mathbb{K}'_0						
11	16, 49, 82, 115, 24, 57, 90, 123, 0, 33, 66, 99	\mathbb{K}_2	₩20						
15	8, 41, 74, 107, 16, 49, 82, 115, 24, 57, 90, 123	\mathbb{K}_3							
19	96, 1, 34, 67, 104, 9, 42, 75, 112, 17, 50, 83	\mathbb{K}_4							
23	120, 25, 58, 91, 96, 1, 34, 67, 104, 9, 42, 75	\mathbb{K}_5	\mathbb{K}'_1						
27	112, 17, 50, 83, 120, 25, 58, 91, 96, 1, 34, 67	\mathbb{K}_6	1						
31	104, 9, 42, 75, 112, 17, 50, 83, 120, 25, 58, 91	\mathbb{K}_7							
35	64, 97, 2, 35, 72, 105, 10, 43, 80, 113, 18, 51	\mathbb{K}_8							
39	88, 121, 26, 59, 64, 97, 2, 35, 72, 105, 10, 43	\mathbb{K}_9	K ′						
43	80, 113, 18, 51, 88, 121, 26, 59, 64, 97, 2, 35	\mathbb{K}_{10}	\mathbb{K}_2'						
47	72, 105, 10, 43, 80, 113, 18, 51, 88, 121, 26, 59	\mathbb{K}_{11}							
51	32, 65, 98, 3, 40, 73, 106, 11, 48, 81, 114, 19	\mathbb{K}_{12}							
55	56, 89, 122, 27, 32, 65, 98, 3, 40, 73, 106, 11	\mathbb{K}'_3							
59	$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$								
63	40, 73, 106, 11, 48, 81, 114, 19, 56, 89, 122, 27	\mathbb{K}_{15}							
67	4, 37, 70, 103, 12, 45, 78, 111, 20, 53, 86, 119	\mathbb{K}_{16}							
71	28, 61, 94, 127, 4, 37, 70, 103, 12, 45, 78, 111	\mathbb{K}_{17}	πε/						
75	20, 53, 86, 119, 28, 61, 94, 127, 4, 37, 70, 103	\mathbb{K}_{18}	\mathbb{K}_4'						
79	12, 45, 78, 111, 20, 53, 86, 119, 28, 61, 94, 127	\mathbb{K}_{19}							
83	100, 5, 38, 71, 108, 13, 46, 79, 116, 21, 54, 87	\mathbb{K}_{20}							
87	124, 29, 62, 95, 100, 5, 38, 71, 108, 13, 46, 79	\mathbb{K}_{21}	\mathbb{K}'_5						
91	556, 89, 122, 27, 32, 65, 98, 3, 40, 73, 106, 11 \mathbb{K}_{13} 948, 81, 114, 19, 56, 89, 122, 27, 32, 65, 98, 3 \mathbb{K}_{14} 340, 73, 106, 11, 48, 81, 114, 19, 56, 89, 122, 27 \mathbb{K}_{15} 74, 37, 70, 103, 12, 45, 78, 111, 20, 53, 86, 119 \mathbb{K}_{16} 728, 61, 94, 127, 4, 37, 70, 103, 12, 45, 78, 111 \mathbb{K}_{17} 7520, 53, 86, 119, 28, 61, 94, 127, 4, 37, 70, 103 \mathbb{K}_{18} 7912, 45, 78, 111, 20, 53, 86, 119, 28, 61, 94, 127 \mathbb{K}_{19} 33100, 5, 38, 71, 108, 13, 46, 79, 116, 21, 54, 87 \mathbb{K}_{20} 57124, 29, 62, 95, 100, 5, 38, 71, 108, 13, 46, 79 \mathbb{K}_{21} 10116, 21, 54, 87, 124, 29, 62, 95, 100, 5, 38, 71 \mathbb{K}_{22} 55108, 13, 46, 79, 116, 21, 54, 87, 124, 29, 62, 95 \mathbb{K}_{23} 9968, 101, 6, 39, 76, 109, 14, 47, 84, 117, 22, 55 \mathbb{K}_{24} 0392, 125, 30, 63, 68, 101, 6, 39, 76, 109, 14, 47 \mathbb{K}_{25} 0784, 117, 22, 55, 92, 125, 30, 63, 68, 101, 6, 39 \mathbb{K}_{26}								
95	108, 13, 46, 79, 116, 21, 54, 87, 124, 29, 62, 95	\mathbb{K}_{23}							
99	68, 101, 6, 39, 76, 109, 14, 47, 84, 117, 22, 55	\mathbb{K}_{24}							
103			πε/						
107	84, 117, 22, 55, 92, 125, 30, 63, 68, 101, 6, 39	\mathbb{K}_{26}	\mathbb{K}_{6}^{\prime}						
111	76, 109, 14, 47, 84, 117, 22, 55, 92, 125, 30, 63								
115	36, 69, 102, 7, 44, 77, 110, 15, 52, 85, 118, 23	\mathbb{K}_{28}							
119	60, 93, 126, 31, 36, 69, 102, 7, 44, 77, 110, 15	\mathbb{K}_{29}	TZ /						
123	52, 85, 118, 23, 60, 93, 126, 31, 36, 69, 102, 7	\mathbb{K}_{30}	\mathbb{K}_{7}'						
127	44,77,110,15,52,85,118,23,60,93,126,31	K21	1						

Table 5. The corresponding bit positions of RK^{r+2} that need to be guessed for detecting the integral property of X^{r+1} at the bit position 4i + 3 ($0 \le i \le 31$).

For example, regarding $(\mathbb{K}_0, \mathbb{K}_1, \mathbb{K}_2, \mathbb{K}_3)$, we have

 $\mathbb{K}_0' = \mathbb{K}_0 \cup \mathbb{K}_1 = \mathbb{K}_0 \cup \mathbb{K}_2 = \mathbb{K}_0 \cup \mathbb{K}_3 = \mathbb{K}_1 \cup \mathbb{K}_2 = \mathbb{K}_1 \cup \mathbb{K}_3 = \mathbb{K}_2 \cup \mathbb{K}_3$

- $= \mathbb{K}_0 \cup \mathbb{K}_1 \cup \mathbb{K}_2 = \mathbb{K}_0 \cup \mathbb{K}_1 \cup \mathbb{K}_3 = \mathbb{K}_1 \cup \mathbb{K}_2 \cup \mathbb{K}_3$
- $= \mathbb{K}_0 \cup \mathbb{K}_1 \cup \mathbb{K}_2 \cup \mathbb{K}_3$
- $= \{0, 33, 66, 99, 8, 41, 74, 107, 16, 49, 82, 115, 24, 57, 90, 123\}.$

11

Thus, if there are b_0 ($b_0 \ge 2$) balanced bits among $(x_3^{r+1}, x_7^{r+1}, x_{11}^{r+1}, x_{15}^{r+1})$, then we can simultaneously use these balanced bits to filter the guessed key bits, whose positions are contained in \mathbb{K}'_0 . In particular, the filtering strength is 2^{-b_0} , i.e., a wrongly guessed key can pass one time filtering with a probability of 2^{-b_0} . Here we define detecting the balanced property for the all remaining guessedkey candidates as one time filtering. For example, assuming all the 4 bits of $(x_3^{r+1}, x_7^{r+1}, x_{11}^{r+1}, x_{15}^{r+1})$ are balanced. At the beginning, there are in total 2^{16} guessed values for the 16 bits of subkey RK^{r+2} whose positions are shown in \mathbb{K}'_0 . For each guessed value, we decrypt $2^d X^{r+3}$ generated by the chosen X^0 to get the information of $(x_3^{r+1}, x_7^{r+1}, x_{11}^{r+1}, x_{15}^{r+1})$, and compute the XOR sum at these 4 bits. If the XOR sum at 4 bits are all 0s, then we keep this guessed value as a candidate, otherwise we discard this guessed value. After 2^{16} times (we call one time filtering), there are 2^{12} remaining candidates in theory. Repeat above process once again, i.e., filtering one more time, there are 2^8 candidates left.

Analyze the filtering. For an r-round distinguisher, we denote b_s the number of balanced bits among the s-th bit group $(x_{16s+3}^{r+1}, x_{16s+7}^{r+1}, x_{16s+11}^{r+1}, x_{16s+15}^{r+1})$ where $0 \leq s \leq 7$. We denote \mathbb{C}_s the set that contains the candidates of the 16 key bits whose positions are in \mathbb{K}'_s for $0 \leq s \leq 7$. Especially, \mathbb{C}_s has in total 2^{16} candidates before the first filtering. In theory, after t_s times filtering for \mathbb{C}_s , $2^{16-t_s \times b_s}$ candidates will survive in \mathbb{C}_s . Particularly, when $t_s \times b_s \geq 16$, the correct value can be recovered. However, it is not consistent with the reality as indicated in the following observation.

Observation 2. For each $0 \le s \le 7$, there are always 2^4 surviving candidates (including the correct value) in \mathbb{C}_s , no matter how many times we filter them.

In order to illustrate this observation more intuitively, we use the case of s = 0 as an example. The related bits of state and subkey for detecting the integral property of $(x_{15}^{r+1}, x_{11}^{r+1}, x_7^{r+1}, x_3^{r+1})$ are simply depicted in Figure 2. Coincidentally, the 12 related bits of X^{r+2} are distributed as the 3 LSBs of 4 Sboxes. Assuming all bits of $(x_{15}^{r+1}, x_{11}^{r+1}, x_7^{r+1}, x_3^{r+1})$ are balanced, then the filtering strength is 2^{-4} and 2^4 candidates will survive in \mathbb{C}_0 after 3 filterings. Let us focus on S_0 . From Property 2, we know that the value of $(x_2^{r+2}, x_1^{r+2}, x_0^{r+2})$ will not be changed if we simultaneously turn over the 4 bits $(y_3^{r+2}, y_2^{r+2}, y_1^{r+2}, y_0^{r+2})$. Note that

$$y_3^{r+2} = rk_{99}^{r+2} \oplus x_{99}^{r+3}, \quad y_2^{r+2} = rk_{66}^{r+2} \oplus x_{66}^{r+2} \\ y_1^{r+2} = rk_{33}^{r+2} \oplus x_{33}^{r+3}, \quad y_0^{r+2} = rk_0^{r+2} \oplus x_0^{r+2}.$$

That is to say, if the correct value of $(rk_{99}^{r+2}, rk_{66}^{r+2}, rk_{33}^{r+2}, rk_0^{r+2})$ is (v_3, v_2, v_1, v_0) , then the guessed value $(v_3 \oplus 1, v_2 \oplus 1, v_1 \oplus 1, v_0 \oplus 1)$ will always survive in the filtering. In other words, there are two values of 4-bit key can pass the filtering for each of the 4 Sboxes, one is the correct value, the other is the negation of the correct one. So, there are in total 2⁴ candidates that will retained in \mathbb{C}_0 .

Observation 2 tells us that we can recover at most 96 bits information of 128bit subkey using integral distinguishers. Then, the remaining 32-bit information can be determined by exhausted searching.

12 Authors Suppressed Due to Excessive Length

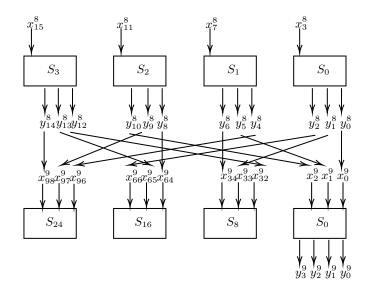


Fig. 2. The related bits of state and subkey for detecting the integral property of $(x_{15}^{r+1}, x_{11}^{r+1}, x_7^{r+1}, x_3^{r+1})$ using 7-round distinguisher, i.e., r = 7.

3.2 Searching for Proper Integral Distinguishers

According to the above analysis, the framework of our (r+2)-round key-recovery attack requires that the *r*-round integral distinguisher has balanced bits at positions 4i+3 ($0 \le i \le 31$) as many as possible. For this, we utilize the MILP-aided tool based on division property, which is the state-of-the-art technique to find integral distinguishers. Here, we omit the MILP modeling details (see [20]), and only illustrate how we obtain the proper integral distinguishers.

For practical attack. Assuming the number of active bits of an integral distinguisher is d, then we need to prepare at least 2^d chosen plaintexts for our attack. To ensure that the attack can be mounted in real time on a PC, we first fix d = 32. With the MILP-based tool, we can efficiently evaluate with d = 32 that the longest integral distinguisher is 8-round and all the 8-round output bits are balanced. Also, there exists no 9-round distinguisher. Note that our framework only focuses on the integral properties of bits whose positions are 4i + 3 ($0 \le i \le 31$), so we can further decrease the number of active bits. As for the positions of active bits, in order to find integral distinguishers fastly, we only try 129 - d cases where the d active bits are continuously arranged on X^0 , i.e., $(x_{d+j-1}^0, \dots, x_j^0)$ for $0 \le j \le 128 - d$ are active bits, instead of enumerating all the $\binom{128}{d}$ trials. In each case, we count the number of balanced bits at positions 4i + 3 for $0 \le i \le 31$, and retain the the one, which has the maximal balanced bits, to mount the practical attack. With this approach, we obtain a 7-round distinguisher with 14 active bits and an 8-round distinguisher with 30 active bits. We denote the two distinguishers by \mathcal{ID}_7 and \mathcal{ID}_8 respectively, and give

their details in Table 6. Note that both of \mathcal{ID}_7 and \mathcal{ID}_8 have 32 balanced bits at positions 4i + 3 ($0 \le i \le 32$).

Table 6. The specification of \mathcal{ID}_7 and \mathcal{ID}_8 . Notice that the bit position '0' is the LSB.

Dist.	Active bits	Number
\mathcal{ID}_7	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	14
\mathcal{ID}_8	$\begin{array}{c}2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,\\18,19,20,21,22,23,24,25,26,27,28,29,30,31\end{array}$	30

For theoretical attack. In order to find the longest integral distinguisher, we straightforwardly set d = 127 and obtain a 13-round distinguisher with 128 balanced bits denoted by \mathcal{ID}_{13} . This result is actually consistent with the evaluation in [1]. Also, there exists no longer distinguisher. Moreover, all the 13-round output bits will become unknown once we decrease the number of active bits. So, for the theoretical attack, we choose the distinguisher \mathcal{ID}_{13} .

3.3 Key-recovery Attacks

In this part, we will utilize the prepared distinguishers to mount two practical and a theoretical key-recovery attacks based on our framework. First, we give some notations. For an *r*-round distinguisher, we denote b_s the number of balanced bits among the *s*-th bit group $(x_{16s+3}^{r+1}, x_{16s+7}^{r+1}, x_{16s+11}^{r+1}, x_{16s+15}^{r+1})$ where $0 \le s \le 7$. We denote \mathbb{C}_s the set that contains the candidates of the 16 key bits whose positions are in \mathbb{K}'_s for $0 \le s \le 7$. At the beginning, \mathbb{C}_s has in total 2^{16} candidates.

A 9-round practical attack. In this attack, we use the 7-round integral disinguisher \mathcal{ID}_7 as shown in Table 6. To be more specific, we have r = 7, d = 14 and $b_s = 4$ for all $0 \leq s \leq 7$. To ensure that there are only 2^4 candidates left in each \mathbb{C}_s , we have to filter each \mathbb{C}_s 3 times. Note that for a fixed s, every time for filtering \mathbb{C}_s requires 2^{14} different chosen plaintexts, but for different s, the chosen plaintexts can be reused. As a result, the data complexity of this attack is $3 \times 2^{14} \approx 2^{15.585}$.

As for the time complexity, we first need to evaluate the cost of one time filtering, which is composed of two parts: partial decryptions and XOR sum computations. In this attack, we regard 9-round BAKSHEESH as a unit time. For each $s \in \{0, 1, \dots, 7\}$, we can see that it takes $16 + 2 \times 4 = 24$ bitwise XORs and 4 lookup-table operations (for the 4-bit inverse Sbox) to get the information of the s-th 4-bit group $(x_{16s+7}^{r+1}, x_{16s+11}^{r+1}, x_{16s+15}^{r+1})$ under a guessed key and known X^{10} . We have tested the latency of a bitwise XOR and a lookup-table in a usual PC. The experimental result shows that the latency of a bitwise XOR is almost equal to that of a lookup-table. For the sake of convenience, we regard a lookup-table as a bitwise XOR. So, the cost of the above partial decryption can be converted to 28 bitwise XORs. In addition, an XOR sum computation at one bit costs $2^{14} - 1$ bitwise XORs under a guessed key. Note that 9-round

14 Authors Suppressed Due to Excessive Length

BAKSHEESH contains 128×10 bitwise XORs and 32×9 lookup-table operations, which can be integrated into 1568 bitwise XORs. Now, let us to compute the 9-round time complexity that composed of the following parts:

- 1. To filter guessed values by integral distinguisher, we need to prepare $2^{14} \times 3 X^{10}$;
- 2. To filter \mathbb{C}_s for each $s \in \{0, \dots, 7\}$ 3 times to get 2^4 candidates, we need $2^{14} \times (2^{16} + 2^{12} + 2^8) \approx 2^{30.093}$ partial decryptions and $2^{16} + 2^{12} + 2^8 \approx 2^{16.093}$ XOR sum computations at 4 bits, which can be transformed to $[2^{30.093} \times 28 + 2^{16.093} \times (2^{14} 1) \times 4] \div 1568 \approx 2^{24.478};$
- 3. To determine the remaining 32-bit information, we need additional 2^{32} 9-round encryptions.

Summing up all the parts, the time complexity to recover 128 key bits is

$$2^{14} \times 3 + 2^{24.478} \times 8 + 2^{32} \approx 2^{32.009}$$

Apparently, the time complexity is mainly caused by filtering the remaining 2^{32} candidates. Both of the time and data complexity of this 9-round attack are practical. Actually, on the platform with i7-8700 CPU @ 3.20GHz and 24 GB RAM, it takes at most 62 minutes to recover all the key bits using single thread.

A 10-round practical attack. In this attack, we use the 8-round distinguisher \mathcal{ID}_8 in Table 6, from which we have r = 8, d = 30 and $b_s = 4$ for each $s \in \{0, \dots, 7\}$. Similarly, we first filter the guessed values by integral distinguisher, and then determing the remaining subkey bits by 10-round encryptions. Therefore, the data complexity of this attack is $3 \times 2^{30} \approx 2^{31.585}$.

However, different from the 9-round attack, we do not have to do partial decryptions under the all 2^{30} 128-bit X^{11} in one time filtering. It is worth mentioning that to get the 4-bit information of $(x_{16s+3}^9, x_{16s+7}^9, x_{16s+11}^9, x_{16s+15}^9)$, we only exploit 16-bit information of X^{11} . In order to decrease the complexity of detecting integral property, we can truncate the values of the 16 bits from X^{11} under 2^{30} 128-bit X^{11} , and save them in a table. The process is described in Algorithm 1. Note that the elements of \mathbb{P} are actually consistent to that of \mathbb{K}'_s . Moreover, the number of elements of \mathbb{T} is at most 2^{16} as there are in total 2^{16} possible values for $(x_{\mathbb{P}[0]}^{11}, \cdots, x_{\mathbb{P}[15]}^{11})$. Namely, the memory complexity to save a table is at most $2^{16} \times 16 = 2^{20}$ bits. Consequently, to detect the integral properties of 2^{30} partial decryptions and b_s XOR sum computations, where an XOR sum computation is derived by $2^{16} - 1$ instead of $2^{30} - 1$ bitwise XORs.

Now, let us compute the time and memory complexity as follows:

- 1. To filter guessed values by integral distinguisher, we need to prepare $2^{30} \times 3 X^{11}$;
- 2. To filter \mathbb{C}_s for each $s \in \{0, \dots, 7\}$ 3 times, we need $2^{16} \times (2^{16} + 2^{12} + 2^8) \approx 2^{32.093}$ partial decryptions and $2^{16} + 2^{12} + 2^8 \approx 2^{16.093}$ XOR sum computations at 4 bits, which can be transformed to $[2^{32.093} \times 28 + 2^{16.093} \times (2^{16} 1) \times 4] \div 1728 \approx 2^{26.338};$

Algorithm 1: Truncating the 16-bit information of X^{11}

Input: The 2^{30} chosen X^0 ; A given list \mathbb{P} that contains 16 bit positions. **Output:** A table contains 16-bit information of X^{11} . 1 Initialize \mathbb{T} as an empty table: **2** for each X^0 do Obtain X^{11} by asking 10-round BAKSHEESH; 3 Truncate 16 bits indexed by \mathbb{P} from X^{11} ; Denote this 16-bit value by $(x_{\mathbb{P}[0]}^{11}, \cdots, x_{\mathbb{P}[15]}^{11})$; /* $\mathbb{P}[\mathbf{i}]$ denotes the *i*-th 4 5 element of \mathbb{P} . */ $\begin{array}{l} \mathbf{if} \ (x_{\mathbb{P}[0]}^{11}, \cdots, x_{\mathbb{P}[15]}^{11}) \ not \ in \ \mathbb{T} \ \mathbf{then} \\ | \ \mathrm{Add} \ (x_{\mathbb{P}[0]}^{11}, \cdots, x_{\mathbb{P}[15]}^{11}) \ \mathrm{into} \ \mathbb{T}; \end{array}$ 6 7 else8 Remove $(x_{\mathbb{P}[0]}^{11}, \cdots, x_{\mathbb{P}[15]}^{11})$ from \mathbb{T} ; 9 10 end 11 end 12 return T.

3. To determine the remaining 32-bit information, we need additional 2^{32} 10-round encryptions.

In summary, the time complexity to recover the 128 key bits is at most

 $2^{30} \times 3 + 2^{26.338} \times 8 + 2^{32} \approx 2^{32.932}.$

Note that each \mathbb{K}'_s $(0 \le s \le 7)$ corresponds to an exclusive table \mathbb{T}_s , which can obtained in parallel by Algorithm 1. Therefore, the memory complexity is at most $2^{20} \times (3 \times 8) = 2^{24.585}$ bits.

A 15-round theoretical attack. We use the 13-round distinguisher, which has 127 active bits and 128 balanced bits, to mount a theoretical 15-round attack. For all $0 \leq s \leq 7$, $b_s = 4$ and we can only filter once for \mathbb{C}_s . The data complexity is 2^{127} chosen plaintexts, the memory complexity is $2^{20} \times 8 = 2^{23}$ bits. As for the time complexity, to filter \mathbb{C}_s by one time, we need at most $2^{16} \times 2^{16} = 2^{32}$ partical decryptions and 2^{16} XOR sum computations at 4 bits, which costs $[2^{32} \times 28 + 2^{16} \times (2^{16} - 1) \times 4] \div 2528 \approx 2^{25.696}$ 15-round encryptions. Note that there are in total $12 \times 8 = 96$ unknown bits for the 128-bit subkey, in which the correct one can be determined by exhausted searching. Therefore, the final time complexity is

$$2^{25.696} \times 8 + 2^{127} + 2^{96} \approx 2^{127.000}.$$

4 Differential and Linear Attacks on Round-reduced BAKSHEESH

In this section, we use SAT models to search for optimal differential and linear trails of BAKSHEESH. We adopt the automated models utilized in previous researches [18], with the results presented in Table 7 and Table 9. It should be noted that we also tried MILP-based automation, but the number of rounds it could achieve was not as high as SAT. Therefore, we ultimately provide the description and analysis based on the SAT model.

4.1 Finding Differential Trails

In our SAT model, we use CNF language to describe the constraints. For the DDT of BAKSHEESH's S-Box, since the DDT values are even, only one extra variable needs to be introduced to describe DDT values: p = 1 for DDT[i][j]=4 and p = 0 for DDT[i][j]=16. As a result, 9 variables are sufficient to describe the propagation rules and corresponding probabilities (8 for input and output differences and 1 for probability), with each S-Box using 27 CNF clauses. Moreover, Matsuis bounding conditions are adopted to accelerate the model-solving process.

The optimal differential bounds for BAKSHEESH are present in Table 7. It should be noted that, although our results align closely with the original paper, the differential bound for 12-round was incorrectly written as 70 instead of 68 in the original paper.

We conducted experimental validation of short-round differential trails, and the probabilities obtained from the experiments matched the search results. Additionally, to verify that there are no errors in the differential trails, we wrote programs to separately compute the output values of the linear and nonlinear layers of the differential trails. The results indicate that the identified differential trails are valid. It is worth mentioning that the 22-round differential trails provided in [1] both showed errors under the aforementioned two verification methods. Moreover, we have also modeled the AddRoundKey operation within

Table 7. Optimal differential bounds for BAKSHEESH (single trail), where p denotes the probability.

$\begin{array}{c} \text{Round} \\ -\log_2 p \end{array}$		-		-	-		-	-	-	
$\begin{array}{c} \text{Round} \\ -\log_2 p \end{array}$	-		-	-		-	-	-		

the aforementioned automated framework, enabling the search for related-key differential trails. The results for the first 9 rounds are shown in Table 8 below, marking the first related-key analysis for BAKSHEESH.

4.2 A 19-round Differential Key-recovery Attack

Considering the 18-round differential trail with a probability of 2^{-120} , as illustrated in Table 7, we present a key-recovery attack on the 19-round BAKSHEESH

17

Table 8. Optimal related-key differential bounds (single trail) for BAKSHEESH.

Round	1	2	3	4	5	6	7	8	9
$-\log_2 p$	0	2	4	6	8	12	16	26	36

cipher by extending the distinguisher with one additional round. The input difference for this distinguisher is (0x0d00940040a00000, 0x00000000000000000), and the output difference is (0x8800890044004c00, 0x2200260011001300). The specific steps for key-recovery are as follows:

- 1. Choose 2¹²¹ pairs of plaintexts with differences equal to the input differences of the 19-round differential trail;
- 2. Filter out incorrect pairs using the inactive bits of the ciphertexts, there are 2^{57} pairs remaining, given that the output difference contains 16 inactive Sboxes;
- 3. Initialize a list of 2^{64} empty counters to guess 64 bits of the subkey RK^{19} ;
- 4. For all 2⁵⁷ pairs, perform a guess-and-filter procedure to identify candidate keys and update the corresponding counters.

We pick the candidate subkey whose counter is maximum as the right subkey. The data complexity of this process is approximately 2^{122} , the memory complexity is 2^{64} and the time complexity is 2^{121} . For the 128-bit master key, we have already determined 64 bits. To recover the remaining 128 - 64 = 64 bits, a brute-force search can be adopted and the time complexity is 2^{64} , while the memory and data complexities are negligible. Then the total time complexity to recovery all the 128-bit key is 2^{121} .

4.3 Finding Linear Trails

In our model, we use CNF language to describe the constraints. For the LAT of BAKSHEESH's S-Box, since the LAT values has three value: 4, -4, 8, we introduce a new variable to describe the corrlation. In total, 9 variables are sufficient to describe the propagation rules and corresponding corrlations, with each S-Box using 36 CNF clauses. Moreover, Matsuis bounding conditions are adopted to accelerate the model-solving process.

We have completed the search for all the optimal linear bounds of BAKSHEESH, the results are presented in Table 9. Given that the known results align with those presented in the original paper, it can be argued that 35 rounds of BAKSHEESH would be sufficient to withstand linear attacks since the the upper linear bound reaches 2^{-64} at the 22 round.

4.4 A 21-round Linear Key-recovery Attack

In the key-recovery attack, we choose to add an extra round after obtaining the linear trail. For an *R*-round attack based on an R-1 round linear trail, the number of plaintexts required is 2^{2c} , where *c* is the correlation of the linear

Table 9. Optimal linear bounds for BAKSHEESH (single trail), where c denotes the correlation.

Round $-\log_2 c$												
$\begin{array}{c} \text{Round} \\ -\log_2 c \end{array}$	-	-		-	-	-	-	-	 -	-	 -	

trail. Denote 4N as the number of key bits that need to be guessed for the additional round. Then the time complexity for partial decryption on ciphertexts is $2^{2c+4N} \div R$, allowing for the recovery of m bits of information, where m is the number of bits which have non-zero output masks. The complexity for exhaustive search of the remaining key bits is 2^{128-m} . To complete the attack, the condition $2c + 4N \le 128 + \log_2(R)$ must be satisfied.

To perform the key-recovery attack, we extended the original SAT model by incorporating the additional round key bits to search for key-recovery-friendly distinguishers. We searched for different parameter sets with this extension, as detailed below.

To perform a 22-round key-recovery, we need to search 21-round trails. In this case, $c \ge 61$, hence $N \le 2$ should be satisfied to surpass brute-force search. Similarly, to perform a 21-round key-recovery, $N \le 4$ needs to be satisfied due to the fact that $c \ge 58$ in this scenario. We searched several sets of parameters to obtain distinguishers that meet the requirements. For example, the parameters (r, c, N) = (21, 62, 2) indicate a 21-round scenario where $c \le 62$ and the key bits to guess is 4N. In addition, we also tested the parameters (r, c, N) = (21, 54, 1), (20, 58, 4), (20, 60, 3), (20, 62, 2), (20, 64, 1).

Based on the above search results, a 22-round attack cannot be completed within this attack framework. Considering the 20-round linear trail with a corrlation of 2^{-60} , we present a key-recovery attack on the 21-round BAKSHEESH cipher by extending the distinguisher with one additional round. The input mask for this distinguisher is (0x090000000140000,0x000000000000000000), and the output mask is (0x500000000000000,0x000000000000000000). The specific steps for key-recovery are as follows:

- 1. The number of plaintext-ciphertext pairs required in the key-recovery is equal to 2^{120} ;
- 2. To complete one round of decryption, 12 bits of the key need to be guessed and 5 bits key information can be obtained. The time complexity of this step is $2^{120} \cdot 2^{12} \div 21 = 2^{127.61}$;
- 3. The exhaustive search complexity for the remaining 123 bits is 2^{123} .

Hence, the time complexity is $2^{127.67}$, the memory complexity is 2^{12} , and the data complexity is 2^{120} .

5 Conclusion

In this paper, we evaluate the security of BAKSHEESH from three perspectives: 1) practical and theorical key-recovery attack based on integral cryptanalysis; 2) a SAT-aided re-evaluation on the security bound against differential cryptanalysis as well as a key-recovery attack; 3) an automated model for linear key-recovery attacks, featuring a 21-round attack. Overall, this paper not only corrects previous inaccuracies but also provides a more detailed cryptographic analysis from both theoretical and practical time perspectives.

Acknowledgement. We would like to thank all the anonymous reviewers for their professional and insightful comments, which help us to significantly improve the quality of this paper. This work was supported by Shandong Provincial Natural Science Foundation (Grant No. ZR2024QA205), the National Natrual Science Foundation of China (Grant No. 62272147, 12471492, 62072161, and 12401687), the Innovation Group Projectof the Natural Science Foundation of Hubei Province of China (Grant No. 2023AFA021), the Science and Technology on Communication Security Laboratory Foundation (Grant No. 6142103012207), the National Key Research and Development Project (Grant No. 2018YFA0704705) and the CAS Project for Young Scientists in Basic Research (Grant No. YSBR-035).

References

- Baksi, A., Breier, J., Chattopadhyay, A., Gerlich, T., Guilley, S., Gupta, N., Hu, K., Isobe, T., Jati, A., Jedlicka, P., Kim, H., Liu, F., Martinasek, Z., Sakamoto, K., Seo, H., Shiba, R., Shrivastwa, R.R.: BAKSHEESH: similar yet different from GIFT. IACR Cryptol. ePrint Arch. p. 750 (2023)
- Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present - towards reaching the limit of lightweight encryption. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017, Taipei, Taiwan, September 25-28, 2017, Proceedings. LNCS, vol. 10529, pp. 321–345. Springer (2017)
- Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR Cryptol. ePrint Arch. p. 404 (2013)
- Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. LNCS, vol. 9815, pp. 123–153. Springer (2016)
- Biham, E.: New types of cryptanalytic attacks using related keys. J. Cryptol. 7(4), 229–246 (1994)
- Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In: Stern, J. (ed.) Advances in Cryptology - EUROCRYPT '99, Prague, Czech Republic, May 2-6, 1999, Proceeding. LNCS, vol. 1592, pp. 12–23. Springer (1999)

- 20 Authors Suppressed Due to Excessive Length
- Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. J. Cryptol. 4(1), 3–72 (1991)
- Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007, Vienna, Austria, September 10-13, 2007, Proceedings. LNCS, vol. 4727, pp. 450–466. Springer (2007)
- 9. Bogdanov, A., Rijmen, V.: Zero-correlation linear cryptanalysis of block ciphers. IACR Cryptol. ePrint Arch. p. 123 (2011)
- Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In: Wang, X., Sako, K. (eds.) Advances in Cryptology - ASIACRYPT 2012, Beijing, China, December 2-6, 2012. Proceedings. LNCS, vol. 7658, pp. 208–225. Springer (2012)
- Daemen, J., Knudsen, L.R., Rijmen, V.: The block cipher square. In: Biham, E. (ed.) Fast Software Encryption, FSE '97, Haifa, Israel, January 20-22, 1997, Proceedings. LNCS, vol. 1267, pp. 149–165. Springer (1997)
- Knudsen, L.R.: Truncated and higher order differentials. In: Preneel, B. (ed.) Fast Software Encryption, Leuven, Belgium, 14-16 December 1994, Proceedings. LNCS, vol. 1008, pp. 196–211. Springer (1994)
- Knudsen, L.R., Wagner, D.A.: Integral cryptanalysis. In: Daemen, J., Rijmen, V. (eds.) Fast Software Encryption, FSE 2002, Leuven, Belgium, February 4-6, 2002, Revised Papers. LNCS, vol. 2365, pp. 112–127. Springer (2002)
- Lai, X., Massey, J.L.: A proposal for a new block encryption standard. In: Damgård,
 I. (ed.) Advances in Cryptology EUROCRYPT '90, Aarhus, Denmark, May 21-24,
 1990, Proceedings. LNCS, vol. 473, pp. 389–404. Springer (1990)
- Matsui, M.: Linear cryptanalysis method for DES cipher. In: Helleseth, T. (ed.) Advances in Cryptology - EUROCRYPT '93, Lofthus, Norway, May 23-27, 1993, Proceedings. LNCS, vol. 765, pp. 386–397. Springer (1993)
- Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-bit blockcipher CLEFIA (extended abstract). In: Biryukov, A. (ed.) Fast Software Encryption, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers. LNCS, vol. 4593, pp. 181–195. Springer (2007)
- Sun, L., Wang, W., Wang, M.: Automatic search of bit-based division property for ARX ciphers and word-based division property. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017, Hong Kong, China, December 3-7, 2017, Proceedings, Part I. LNCS, vol. 10624, pp. 128–157. Springer (2017)
- Sun, L., Wang, W., Wang, M.: Accelerating the search of differential and linear characteristics with the SAT method. IACR Trans. Symmetric Cryptol. 2021(1), 269–315 (2021)
- Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. LNCS, vol. 9056, pp. 287–314. Springer (2015)
- Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology - ASIACRYPT 2016, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I. LNCS, vol. 10031, pp. 648–678 (2016)