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

Takanori Isobe and Kyoji Shibutani

Sony Corporation
1-7-1 Konan, Minato-ku, Tokyo 108-0075, Japan
{Takanori.Isobe,Kyoji.Shibutani}@jp.sony.com

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

**Keywords:** block cipher, meet-in-the-middle attack, key schedule, CAST-128, SHACAL-2, KATAN, FOX128, Blowfish, all subkeys recovery attack.

## 1 Introduction

Meet-in-the-middle (MITM) attack, originally introduced in [9], is a generic cryptanalytic technique for block ciphers. It was extended to preimage attacks on hash functions, and several novel techniques to extend the attack have been developed [3,14,4,25,5,11]. Then, it has been shown that those advanced techniques are also applied to block ciphers [7,13,6,18].

Since the MITM attack mainly exploits a low key-dependency in a key schedule, it works well for a block cipher having a simple key schedule such as a key schedule based only on a bit permutation. In fact, most of the recent MITM attacks were applied to ciphers having a simple key schedule such as KTANTAN, GOST, IDEA, XTEA, LED and Piccolo [7,13,18,26,15]. However, as far as we know, no results have been known so far for block ciphers consisting of a complex

L.R. Knudsen and H. Wu (Eds.): SAC 2012, LNCS 7707, pp. 202–221, 2013.

key schedule[1] except for the recent attack on AES [6]. In general, the MITM attack at least requires two sets of neutral key bits, which are parts of secret key bits, to compute two functions independently. For a simple key schedule such as a permutation based key schedule, since each subkey is directly derived from some secret key bits, it is relatively simple to find "good" neutral key bits. Yet, for a cipher equipped with a complex key schedule, finding neutral key bits, which is the core technique of the MITM attack, is likely to be complicated and specific. For instance, the MITM attack on the 8-round reduced AES-128 in [6] looks complicated and specific, i.e., it seems difficult to directly apply their technique to other block ciphers not having the AES key schedule.

In this paper, we extend the MITM attack, then present a generic and simple approach to evaluate the security of ciphers employing a complex key schedule against the MITM attack. The basic concept of our approach is converting the game of finding the user-provided key (or the secret key) to the game of finding all subkeys so that the analysis can be independent from the structure of the key schedule, by regarding all subkeys as independent variables. This simple conversion enables us to apply the MITM attack to a cipher using any key schedule without analyzing the key schedule. We refer this attack as all subkeys recovery (ASR) attack for simplicity, while our approach is a variant of the MITM attack. We first apply the ASR attack to CAST-128, Blowfish and FOX128 in a straightforward way, then show the best attacks on them with respect to the number of attacked rounds in literature. Moreover, to construct more efficient attack, we present how to efficiently find useful state that contains a smaller number of subkey bits by analyzing internal components, then apply it to SHACAL-2 and KATAN family. The attacks presented in this paper are summarized in Table 6 (see also Table 2). We emphasize that our attack can be applied to any block cipher having any key schedule function even if the key schedule is an ideally random function. This implies that our approach gives generic lower bounds on the security of several block ciphers against the MITM attacks.

This paper is organized as follows. Section 2 gives some notations used throughout this paper. The basic concept of the ASR attack is presented in Section 3. The applications of the basic ASR attack to CAST-128 and Blowfish are demonstrated in Section 4. Some advanced techniques and those applications to SHACAL-2 and KATAN family are introduced in Sections 5 and 6, respectively. Section 7 discusses several features of the ASR attack. Finally, we conclude in Section 8.

## 2 Notation

The following notation will be used throughout this paper:

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

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

## 3 All Subkeys Recovery (ASR) Attack

In this section, the basic concept of the all subkeys recovery (ASR) attack is introduced. First, we briefly present the basic concept, then, the detailed procedure of the basic ASR attack is given. Finally, we show an ASR attack on the balanced Feistel network as a concrete example.

### 3.1 Basic Concept

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

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

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

Moreover, in the ASR attack, the underlying key schedule can be treated as an ideal function. In other words, our attack works even if the key schedule is an ideal function. A concrete block cipher employs an weaker key schedule than an ideal one. Thus, the number of attacked rounds may be extended by thoroughly analyzing the key schedule. However, even without analyzing the key schedule, we show several best attacks on several block ciphers in the single-key setting in the following sections. Thanks to the MITM approach, our attack requires extremely low data requirement, though it requires a lot of memory which is the same order as the time complexity. We may employ memoryless collision search [23] to reduce the memory requirements, while it requires a slightly larger computations, i.e., the time complexity is multiplied by a small constant.

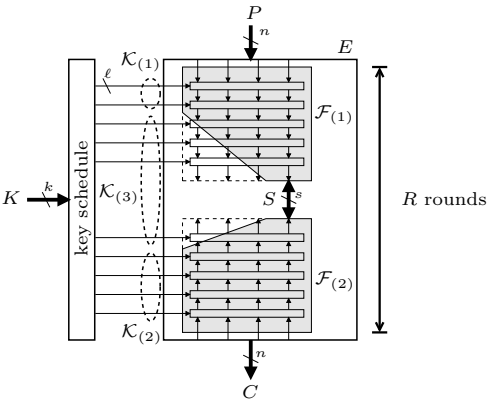


**Fig. 1.** Basic concept of ASR attack

### 3.2 Recovering All Subkeys by MITM Approach (Basic Attack)

Let us explain the basic procedure of the ASR attack. As explained in the previous section, we regard all subkeys in the cipher as independent variables. Then, we apply the MITM approach to determine all subkeys that map a plaintext to the ciphertext encrypted by the (unknown) secret key. Suppose that $n$-bit block cipher $E$ accepting a $k$-bit secret key $K$ consists of $R$ rounds, and an $\ell$-bit subkey is introduced each round (see Fig. 1).

First, an attacker determines an $s$-bit matching state $S$. The state $S$ can be computed from a plaintext $P$ and a set of subkey bits $\mathcal{K}_{(1)}$ by a function $\mathcal{F}_{(1)}$ as $S = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$. Similarly, $S$ can be computed from the ciphertext $C$ and another set of subkey bits $\mathcal{K}_{(2)}$ by a function $\mathcal{F}_{(2)}$ as $S = \mathcal{F}^{-1}_{(2)}(C, \mathcal{K}_{(2)})$. $\mathcal{K}_{(3)}$ denotes a set of the remaining subkey bits, i.e., $|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}| + |\mathcal{K}_{(3)}| = R \cdot \ell$. By using those $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$, the attacker can independently compute $\mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$ and $\mathcal{F}^{-1}_{(2)}(C, \mathcal{K}_{(2)})$. Note that the equation $\mathcal{F}_{(1)}(P, \mathcal{K}_{(1)}) = \mathcal{F}^{-1}_{(2)}(C, \mathcal{K}_{(2)})$ holds when the guessed subkey bits $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are correct. Due to parallel guesses of $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$, we can efficiently check if the guessed key bits are correct. After this process, it is expected that there will be $2^{R \cdot \ell - s}$ key candidates. Note that the number of key candidates can be reduced by parallel performing the matching with additional plaintext/ciphertext pairs. In fact, using $N$ plaintext/ciphertext pairs, the number of key candidates is reduced to $2^{R \cdot \ell - N \cdot s}$, as long as $N \leq (|\mathcal{K}_{(1)}| + |\mathcal{K}_{(2)}|)/s$. Finally, the attacker exhaustively searches the correct key from the surviving key candidates. The required computations (i.e. the number of encryption function calls) of the attack in total $C_{comp}$ is estimated as

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

The number of required plaintext/ciphertext pairs is $\max(N, \lceil (R \cdot \ell - N \cdot s)/n \rceil)$. The required memory is about $\min(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N$ blocks, which is the cost of the table used for the matching. Obviously, the ASR attack works faster than

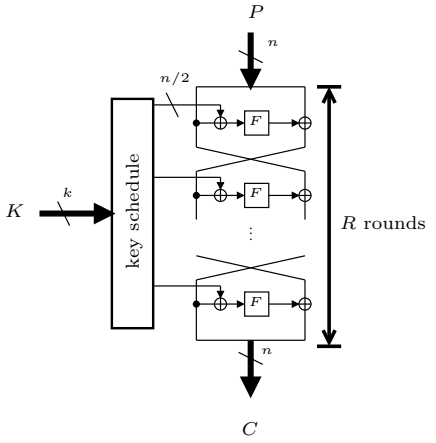


**Fig. 2.** Balanced Feistel network

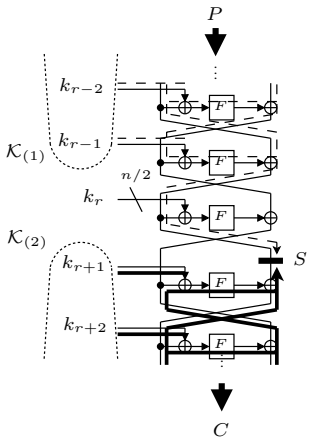


**Fig. 3.** Partial matching

the brute force attack when Eq.(1) is less than $2^k$, which is required computations for the brute force attack. For simplicity, we omit the cost of memory access for finding a match between two lists, assuming that the time complexity of one table look-up is negligible compared to that of one computation of $\mathcal{F}_{(1)}$ or $\mathcal{F}_{(2)}$. The assumption is quite natural in most cases. However, strictly speaking, those costs should be considered. In other words, the cost of $(\max(2^{|\mathcal{K}_{(1)}|}, 2^{|\mathcal{K}_{(2)}|}) \times N)$ memory accesses is added to Eq.(1).

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

### 3.3 ASR Attack on Balanced Feistel Networks

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

As depicted in Fig. 3, an $(n/2)$-bit state $S$ can be computed independently from $(n/2)$-bit subkey $k_r$. In other words, $S$ can be computed from $P$ and $\mathcal{K}_{(1)} \in \{k_1, k_2, ..., k_{r-1}\}$. Also, $S$ can be obtained from $C$ and $\mathcal{K}_{(2)} \in \{k_{r+1}, k_{r+2}, ..., k_R\}$.

When $n = k/2$ (e.g., a 128-bit block cipher accepting a 256-bit key), 7 rounds of $E$ can be attacked in a straightforward manner. In this attack, both $\mathcal{F}_{(1)}$ and $\mathcal{F}_{(2)}$ are composed of 3 rounds of $E$, and thus the sizes of $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$ are both $3n/2$ bits. As explained in Section 3.2, using six plaintext/ciphertext pairs, the total time complexity $C_{comp}$ is estimated as

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

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

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

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

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

Roughly speaking, when Eq.(1) is less than $2^k$, the ASR attack works faster than the brute force attack. Therefore, the necessary condition for the basic ASR attack is that the size of subkey is less than the size of secret key.

## 4 Basic ASR Attacks on CAST-128 and Blowfish

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

The basic parameters of the ciphers analyzed in this paper are listed in Table 1. Table 2 shows the parameters of our (ASR) attacks in this paper.

### 4.1 Descriptions of CAST-128 and Blowfish

**Description of CAST-128.** CAST-128 [1,2] is a 64-bit Feistel block cipher accepting a variable key size from 40 up to 128 bits (but only in 8-bit increments). The number of rounds is 16 when the key size is longer than 80 bits. First, the algorithm divides the 64-bit plaintext into two 32-bit words $L_0$ and $R_0$, then the $i$-th round function outputs two 32-bit data $L_i$ and $R_i$ as follows:

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

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

**Table 1.** Basic parameters of our target ciphers

| algorithm | block size ($n$) | key size ($k$) | subkey size ($\ell$) | # rounds ($R$) |
|---|---|---|---|---|
| CAST-128 [1] | 64 | $40 \leq k \leq 128$ | 37 | 12 ($k \leq 80$), 16 ($k > 80$) |
| Blowfish [27] | 64 | $128 \leq k \leq 448$ | 32 | 16 |
| Blowfish-8R* [27] | 64 | $128 \leq k \leq 192$ | 32 | 8 |
| SHACAL-2 [12] | 256 | $k \leq 512$ | 32 | 64 |
| KATAN32/48/64 [8] | 32/48/64 | 80 | 2 | 254 |
| FOX128 [16] | 128 | 256 | 128 | 16 |

$*$ Fewer iteration version of Blowfish specified in [27] as a possible simplification.

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

**Description of Blowfish.** Blowfish [27] is a 16-round Feistel block cipher with 64-bit block and variable key size from 128 up to 448 bit. Several possible simplifications of Blowfish were also described in [27]. We refer one of them that is an 8-round variant (fewer iterations) of Blowfish accepting less than 192 bits of key as Blowfish-8R. First, Blowfish divides a 64-bit plaintext into two 32-bit state $L_0$ and $R_0$. Then the $i$-th round output state $(L_i||R_i)$ is computed as follows:

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

where $K_i^{rnd}$ is a 32-bit round key at round $i$. The F-function $F$ consists of four $8 \times 32$ key dependent S-boxes. Note that, in the last round, the 64-bit round key is additionally XORed with the 64-bit state. In this paper, we assume that F-function is known by an attacker, and it is a random 32-bit function. The assumption, i.e., the known F-function setting, has already been used in [29,17].

### 4.2 ASR Attack on 7-Round Reduced CAST-128

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

**Table 2.** Parameters of our attacks presented in this paper

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

† Known F-function setting.

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

The number of required known plaintext/ciphertext pairs is only 6 (= max(6, $\lceil(258-6\cdot32)/64\rceil$), and the required memory is about $2^{114}$ (= $\min(2^{111}, 2^{111})\times 6$) blocks. Recall that our attack works faster than the exhaustive key search only when the key size of the reduced CAST-128 is more than 114 bits. As far as we know, the previous best attack on the reduced CAST-128 was for only 6 rounds in the single-key setting [31][2]. Thus, surprisingly, this simple attack exceeds the previously best attack on the reduced CAST-128 with respect to the number of attacked rounds.

### 4.3 ASR Attack on Full Blowfish in Known F-function Setting

In this section, we apply the ASR attack on Blowfish block cipher in the known F-function setting as with [29,17].

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

$$C_{comp} = \max(2^{256}, 2^{288}) \times 9 + 2^{576-9\cdot32} = 2^{292}.$$

The number of required data is only 9 (=max(9, $\lceil(576-9\cdot32)/64\rceil$)) known plaintext/ciphertext pairs, and required memory is about $2^{260}$ (=$\min(2^{256}, 2^{288}) \times 9$) blocks. In this setting, the attack works faster than the exhaustive key search when the key size is more than 292 bits.

Similarly to the attack on the full Blowfish, for the full (8-round) Blowfish-8R, we choose $R_4$ as the 32-bit matching state, i.e., $s = 32$. Then $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$

[2] The differential attacks on the 8- and 9-round reduced CAST-128 in weak-key setting were presented in [30].

include 128 (= 32 × 4) and 160 (= 32 × 5) bits of subkeys, respectively, and $\mathcal{K}_{(3)} = 32$. When $N = 5$ ($\leq (128 + 160)/32$), the time complexity for finding all subkeys is estimated as

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

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

Note that these attacks are the first results on the full Blowfish with all key classes in the known F-function setting, while the attacks presented in [29,17] work only in the weak key setting, i.e., weak F-functions.

## 5 Application to SHACAL-2

In this section, we apply the ASR attack to SHACAL-2 block cipher. After a brief description of SHACAL-2, we present the basic ASR attack on the reduced SHACAL-2 without analyzing the internal functions of the cipher. Then, by analyzing the functions of the cipher, we show the matching state that contains fewer bits of subkeys. Finally, we demonstrate an advanced ASR attack by using this matching state. Recall that the basic ASR attack regards internal components such as an F-function of the balanced Feistel network as a black-box function, while the advanced ASR attack analyzes the internal components. An ASR attack on the 5-round reduced FOX128 is presented in Appendix A as another example of the advanced ASR attack.

### 5.1 Description of SHACAL-2

SHACAL-2 [12] is a 256-bit block cipher based on the compression function of SHA-256 [10]. It was submitted to the NESSIE project and selected to be in the NESSIE portfolio [22].

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

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

where $K_i$ is the $i$-th step constant, $W_i$ is the $i$-th step key (32-bit), and the functions $Ch$, $Maj$, $\Sigma_0$ and $\Sigma_1$ are given as follows:

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

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

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

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

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

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

### 5.2 Basic ASR Attack on 37-Step Reduced SHACAL-2

We directly apply the ASR attack described in Section 3 to SHACAL-2. This leads to the attack on the 37-step reduced SHACAL-2.

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

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

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

Surprisingly, this simple attack exceeds the previous best attack on the reduced SHACAL-2 in the single-key setting for 32 steps [28] with respect to the

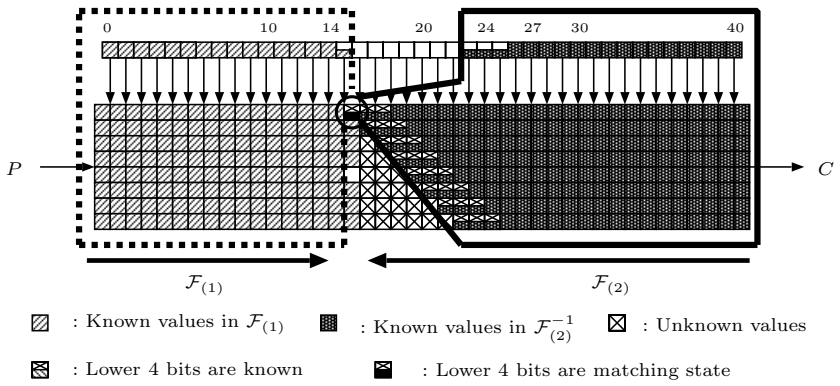


**Fig. 4.** Overview of 41 round attack on reduced SHACAL-2

number of attacked rounds [3]. In the following, by deeply analyzing the functions used in SHACAL-2, we show further improvements.

### 5.3 Advanced ASR Attack on 41-Step Reduced SHACAL-2

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

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

**Backward Computation in $\mathcal{F}_{(2)}$:** We give the following observation.

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

[3] The MITM preimage attacks on the reduced SHA-256 were proposed in [5,19]. However, these attacks can not be directly applied to SHACAL-2, because in the block cipher setting, these attacks require the code book, i.e., they require all plaintext/ciphertext pairs. Also, due to those high time complexity, they do not seem to work faster than the exhaustive key search.

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

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

Thus, $p_{j-1}$ except for $H_{j-1}$ is obtained from $p_j$. The lower $t$ bits of $H_{j-1}$ can be computed from $p_j$ and the lower $t$ bits of $W_{j-1}$. Similarly, from $A_{j-1}, ..., G_{j-1}$ and the lower $t$ bits of $H_{j-1}$ and $W_{j-2}$, we can compute $A_{j-2}, ..., F_{j-2}$ and the lower $t$ bits of $G_{j-2}$ and $H_{j-2}$. Furthermore, $A_{j-3}, ..., E_{j-3}$ and the lower $t$ bits of $F_{j-3}$ and $G_{j-3}$ are computed from $A_{j-2}, ..., F_{j-2}$ and the lower $t$ bits of $G_{j-2}$, $H_{j-2}$ and $W_{j-3}$. Again, as mentioned in [14], $A_{j-10}$ is determined by $p_{j-3}$ without $W_{j-10}, ..., W_{j-16}$. This relation can be translated to "the lower $t$ bits of $A_{j-10}$ are determined from only $A_{j-3}, ..., E_{j-3}$ and the lower $t$ bits of $F_{j-3}$, $G_{j-3}$ and $H_{j-3}$". Therefore, $A_{j-10}$ can be obtained from $p_j$ and the lower $t$ bits of $W_{j-1}, W_{j-2}$ and $W_{j-3}$.

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

**Evaluation.** Recall that the matching state $S$ is the lower 4 bits of $A_{16}$, $|\mathcal{K}_{(1)}| = 484$, $|\mathcal{K}_{(2)}| = 492$ and $|\mathcal{K}_{(3)}| = 336$ $(= 32 \times 7 + (32-4) \times 4)$. Thus, using 244 known plaintext/ciphertext pairs (i.e. $N = 244 \leq (484 + 492)/4$), the time complexity for finding all subkeys is estimated as

$$C_{comp} = \max(2^{484}, 2^{492}) \times 244 + 2^{1312 - 244 \cdot 4} = 2^{500}.$$

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

## 6 Application to KATAN Family

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

**Table 3.** Parameters of KATAN family

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

### 6.1 Description of KATAN

KATAN [8] family is a feedback shift register-based block cipher consisting of three variants: KATAN32, KATAN48 and KATAN64 whose block sizes are 32-, 48- and 64-bit, respectively. All of the KATAN ciphers use the same key schedule accepting an 80-bit key and 254 rounds. The plaintext is loaded into two shift registers $L_1$ and $L_2$. Each round, $L_1$ and $L_2$ are shifted by one bit, and the least significant bits of $L_1$ and $L_2$ are updated by $f_b(L_2)$ and $f_a(L_1)$, respectively. The bit functions $f_a$ and $f_b$ are defined as follows:

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

where $L[x]$ denotes the $x$-th bit of $L$, $IR$ denotes the round constant, and $k_{2i}$ and $k_{2i+1}$ denote the 2-bit $i$-th round key. Note that for KATAN family, the round number starts from 0 instead of 1, i.e., KATAN family consists of round functions starting from the 0-th round to the 253-th round. $L_1^i$ or $L_2^i$ denote the $i$-th round registers $L_1$ or $L_2$, respectively. For KATAN48 or KATAN64, in each round, the above procedure is iterated twice or three times, respectively. All of the parameters for the KATAN ciphers are listed in Table 3.

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

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

### 6.2 Attack Strategy

Recall that the key of our attack is to find the state that contains as small number of subkey bits as possible. In order to find such states, we exhaustively observe the number of key bits involved in each state per round. A pseudo code for counting the number of subkey bits involved in the forward direction for KATAN32 is described in Algorithm 1. We use similar code to observe how many subkey bits are affected to each state of KATAN32 in the backward direction. Also, similar codes are used for counting the number of subkey bits related to each state of KATAN48 and KATAN64. As an example, Table 4 shows the results obtained by this algorithm when $R = 63$ of KATAN32 in the forward direction.

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

**Require:** $R$ /* Evaluated number of rounds */
**Ensure:** $\mathcal{LK}_1[0], \ldots, \mathcal{LK}_1[|L_1|-1]$ and $\mathcal{LK}_2[0], \ldots, \mathcal{LK}_2[|L_2|-1]$ /* The number of key bits involved in each state after $R$ round */

$\mathcal{LK}_1[i] \leftarrow 0$ for $i = 0, \ldots, |L_1|-1$
$\mathcal{LK}_2[i] \leftarrow 0$ for $i = 0, \ldots, |L_2|-1$
**for** $i = 0$ to $R-1$ **do**
  **for** $j = 0$ to $|L_1|-2$ **do**
    $\mathcal{LK}_1[(|L_1|-1)-j] \leftarrow \mathcal{LK}_1[(|L_1|-1)-j-1]$
  **end for**
  **for** $j = 0$ to $|L_2|-2$ **do**
    $\mathcal{LK}_2[(|L_2|-1)-j] \leftarrow \mathcal{LK}_2[(|L_2|-1)-j-1]$
  **end for**
  $\mathcal{LK}_1[0] \leftarrow \mathcal{LK}_2[y_1] + \mathcal{LK}_2[y_2] + \mathcal{LK}_2[y_3] + \mathcal{LK}_2[y_4] + \mathcal{LK}_2[y_5] + \mathcal{LK}_2[y_6] + 1$
  $\mathcal{LK}_2[0] \leftarrow \mathcal{LK}_1[x_1] + \mathcal{LK}_1[x_2] + \mathcal{LK}_1[x_3] + \mathcal{LK}_1[x_4] + (\mathcal{LK}_1[x_5] \cdot IR) + 1$
**end for**
**return** $\mathcal{LK}_1[0], \ldots, \mathcal{LK}_1[|L_1|-1]$ and $\mathcal{LK}_2[0], \ldots, \mathcal{LK}_2[|L_2|-1]$

### 6.3 ASR Attack on 110-Round Reduced KATAN32

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

**Forward Computation in $\mathcal{F}_{(1)}$:** As shown in Table 4, $L_2^{63}[18]$ depends on 68 subkey bits. This implies that $L_2^{63}[18]$ can be computed by a plaintext $P$ and 68 bits of subkeys. More specifically, $L_2^{63}[18] = \mathcal{F}_{(1)}(P, \mathcal{K}_{(1)})$, where $\mathcal{K}_{(1)} \in \{k_0, ..., k_{54}, k_{56}, k_{57}, k_{58}, k_{60}, ..., k_{64}, k_{68}, k_{71}, k_{73}, k_{77}, k_{88}\}$ and $|\mathcal{K}_{(1)}| = 68$.

**Backward Computation in $\mathcal{F}_{(2)}$:** Table 5 shows the result obtained by Algorithm 1 modified to backward direction on KATAN32 with $R = 47$ starting from 110 round. In the backward computation, the matching state $L_2^{63}[18]$ is computed as $L_2^{63}[18] = \mathcal{F}_{(2)}^{-1}(C, \mathcal{K}_{(2)})$, where $\mathcal{K}_{(2)} \in \{k_{126}, k_{138}, k_{142}, k_{146}, k_{148}, k_{150}, k_{153}, k_{154}, k_{156}, k_{158}, k_{160}, \ldots k_{219}\}$, and $|\mathcal{K}_{(2)}| = 70$.

**Evaluation.** For the 110-round reduced KATAN32, the matching state $S$ is chosen as $L_2^{63}[18]$ (1-bit state). Since $|\mathcal{K}_{(3)}| = 82 (= 2 \times 110 - 68 - 70)$ which is more than 80 bits, we first determine only $\mathcal{K}_{(1)}$ and $\mathcal{K}_{(2)}$. After that, we additionally mount the MITM approach in order to determine the remaining 82 bits of subkeys.

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

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

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

**Table 4.** Results on KATAN32 with $R = 63$ in forward direction (starting round = 0)

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

**Table 5.** Results on KATAN32 with $R = 47$ in backward direction (starting round = 109)

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

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

### 6.4 ASR Attack on 100-Round Reduced KATAN48

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

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

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

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

### 6.5 ASR Attack on 94-Round Reduced KATAN64

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

**Table 6.** Summary of the attacks in the single-key setting

| algorithm | # attacked rounds | time | memory [block] | data | reference |
|---|---|---|---|---|---|
| CAST-128 | 6 | $2^{88.51}$ | Not given | $2^{53.96}$ KP | [31] |
| | 7 | $2^{114}$ | $2^{114}$ | 6 KP | this paper (Section 4.2) |
| Blowfish$^{*1}$ | 4 | - | - | $2^{21}$ CP | [24] |
| Blowfish$^{\dagger}$ | 8 | - | - | $2^{48}$ CP | [29] |
| | 16 | $2^{292}$ | $2^{260}$ | 9 KP | this paper (Section 4.3) |
| Blowfish-8R$^{\dagger}$ | 8 | $2^{160}$ | $2^{131}$ | 5 KP | this paper (Section 4.3) |
| SHACAL-2 | 32 | $2^{504.2}$ | $2^{48.4}$ | $2^{43.4}$ CP | [28] |
| | 41 | $2^{500}$ | $2^{492}$ | 244 KP | this paper (Section 5) |
| KATAN32$^{*2}$ | 78 | $2^{76}$ | Not given | $2^{16}$ CP | [21] |
| | 110 | $2^{77}$ | $2^{75.1}$ | 138 KP | this paper (Section 6.3) |
| KATAN48$^{*2}$ | 70 | $2^{78}$ | Not given | $2^{31}$ CP | [21] |
| | 100 | $2^{78}$ | $2^{78}$ | 128 KP | this paper (Section 6.4) |
| KATAN64$^{*2}$ | 68 | $2^{78}$ | Not given | $2^{32}$ CP | [21] |
| | 94 | $2^{77.68}$ | $2^{77.68}$ | 116 KP | this paper (Section 6.5) |
| FOX128 | 5 | $2^{205.6}$ | Not given | $2^{9}$ CP | [32] |
| | 5 | $2^{228}$ | $2^{228}$ | 14 KP | this paper (Appendix A) |

† Known F-function setting.

∗1 The attacks on the full Blowfish in the weak key setting were presented in [29] and [17].

∗2 The accelerating key search techniques for the full KATAN32/48/64 were presented in [20].

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

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

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

## 7 Discussion

On the ASR attack, an attacker attempts to recover all subkeys instead of the user-provided key, regarding all subkeys as independent variables. Note that, if there is no equivalent key, which is a reasonable assumption for a moderate block

cipher, there obviously exist unique subkeys that map a given plaintext to the ciphertext encrypted by a secret key. In the standard MITM attack, determining neutral key bits and constructing initial structure called bicliques seem two of the most complicated and important parts in the attack process. However, in our attack, those two procedures are not required, since the attacker focuses only on subkeys and all subkeys are treated equally. Moreover, in the ASR attack, it is not mandatory to analyze the underlying key schedule, since it basically focuses only on the data processing part. These features make the attack simple and generic. While the ASR attack is simple and generic as explained, it is still powerful attack. Indeed, we can significantly improve the previous results on several block ciphers as summarized in Table 6 (see also Table 2 for the details of the target ciphers). We emphasize that our approach is not polynomial-advantage attack such as [20], which requires access of all possible keys, but exponential-advantage attack. Moreover, our attack works on the block ciphers using any key schedule functions even if it is ideal.

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

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

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

## 8 Conclusion

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

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

**Acknowledgments.** The authors would like to thank to the anonymous referees for their fruitful comments and suggestions.

## References

1. Adams, C.: The CAST-128 encryption algorithm. RFC-2144 (May 1997)
2. Adams, C.: Constructing symmetric ciphers using the CAST design procedure. Des. Codes Cryptography 12(3), 283–316 (1997)
3. Aoki, K., Sasaki, Y.: Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 103–119. Springer, Heidelberg (2009)
4. Aoki, K., Sasaki, Y.: Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 70–89. Springer, Heidelberg (2009)
5. Aoki, K., Guo, J., Matusiewicz, K., Sasaki, Y., Wang, L.: Preimages for Step-Reduced SHA-2. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 578–597. Springer, Heidelberg (2009)
6. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: Lee, D.H. (ed.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 344–371. Springer, Heidelberg (2011)
7. Bogdanov, A., Rechberger, C.: A 3-Subset Meet-in-the-Middle Attack: Cryptanalysis of the Lightweight Block Cipher KTANTAN. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 229–240. Springer, Heidelberg (2011)
8. De Cannière, C., Dunkelman, O., Knežević, M.: KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 272–288. Springer, Heidelberg (2009)
9. Diffie, W., Hellman, M.E.: Exhaustive cryptanalysis of the NBS Data Encryption Standard. IEEE Computer 10, 74–84 (1977)
10. FIPS: Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180-4
11. Guo, J., Ling, S., Rechberger, C., Wang, H.: Advanced Meet-in-the-Middle Preimage Attacks: First Results on Full Tiger, and Improved Results on MD4 and SHA-2. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 56–75. Springer, Heidelberg (2010)
12. Handschuh, H., Naccache, D.: SHACAL. NESSIE Proposal (updated) (October 2001), `https://www.cosic.esat.kuleuven.be/nessie/updatedPhase2Specs/SHACAL/shacal-tweak.zip`
13. Isobe, T.: A Single-Key Attack on the Full GOST Block Cipher. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 290–305. Springer, Heidelberg (2011)
14. Isobe, T., Shibutani, K.: Preimage Attacks on Reduced Tiger and SHA-2. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 139–155. Springer, Heidelberg (2009)
15. Isobe, T., Shibutani, K.: Security Analysis of the Lightweight Block Ciphers XTEA, LED and Piccolo. In: Susilo, W., Mu, Y., Seberry, J. (eds.) ACISP 2012. LNCS, vol. 7372, pp. 71–86. Springer, Heidelberg (2012)
16. Junod, P., Vaudenay, S.: FOX: A New Family of Block Ciphers. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 114–129. Springer, Heidelberg (2004)

17. Kara, O., Manap, C.: A New Class of Weak Keys for Blowfish. In: Biryukov, A. (ed.) FSE 2007. LNCS, vol. 4593, pp. 167–180. Springer, Heidelberg (2007)
18. Khovratovich, D., Leurent, G., Rechberger, C.: Narrow-Bicliques: Cryptanalysis of Full IDEA. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 392–410. Springer, Heidelberg (2012)
19. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 244–263. Springer, Heidelberg (2012)
20. Knellwolf, S.: Meet-in-the-middle cryptanalysis of KATAN. In: Proceedings of the ECRYPT Workshop on Lightweight Cryptography (2011)
21. Knellwolf, S., Meier, W., Naya-Plasencia, M.: Conditional Differential Cryptanalysis of NLFSR-Based Cryptosystems. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 130–145. Springer, Heidelberg (2010)
22. NESSIE consortium: NESSIE portfolio of recommended cryptographic primitives (2003), `https://www.cosic.esat.kuleuven.be/nessie/deliverables/decision-final.pdf`
23. van Oorschot, P.C., Wiener, M.J.: Parallel collision search with cryptanalytic applications. J. Cryptology 12(1), 1–28 (1999)
24. Rijmen, V.: Cryptanalysis and design of iterated block ciphers. Doctoral Dissertation, K. U. Leuven (1997)
25. Sasaki, Y., Aoki, K.: Finding Preimages in Full MD5 Faster Than Exhaustive Search. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 134–152. Springer, Heidelberg (2009)
26. Sasaki, Y., Wang, L., Sakai, Y., Sakiyama, K., Ohta, K.: Three-Subset Meet-in-the-Middle Attack on Reduced XTEA. In: Mitrokotsa, A., Vaudenay, S. (eds.) AFRICACRYPT 2012. LNCS, vol. 7374, pp. 138–154. Springer, Heidelberg (2012)
27. Schneier, B.: Description of a New Variable-length Key, 64-bit Block Cipher (Blowfish). In: Anderson, R. (ed.) FSE 1993. LNCS, vol. 809, pp. 191–204. Springer, Heidelberg (1994)
28. Shin, Y., Kim, J.-S., Kim, G., Hong, S.H., Lee, S.-J.: Differential-Linear Type Attacks on Reduced Rounds of SHACAL-2. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 110–122. Springer, Heidelberg (2004)
29. Vaudenay, S.: On the Weak Keys of Blowfish. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 27–32. Springer, Heidelberg (1996)
30. Wang, M., Wang, X., Chow, K.P., Hui, L.C.K.: New differential cryptanalytic results for reduced-round CAST-128. IEICE Transactions 93-A(12), 2744–2754 (2010)
31. Wang, M., Wang, X., Hu, C.: New Linear Cryptanalytic Results of Reduced-Round of CAST-128 and CAST-256. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 429–441. Springer, Heidelberg (2009)
32. Wu, W., Zhang, W., Feng, D.: Integral Cryptanalysis of Reduced FOX Block Cipher. In: Won, D.H., Kim, S. (eds.) ICISC 2005. LNCS, vol. 3935, pp. 229–241. Springer, Heidelberg (2006)

# Appendix

## A Application to FOX128

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

### A.1 Description of FOX128

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

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

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

### A.2 ASR Attack on 5-Round Reduced FOX128

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

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

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

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

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