Mathematical and Computer Modelling 55 (2012) 26–34

Contents lists available at SciVerse ScienceDirect

Mathematical and Computer Modelling

journal homepage: www.elsevier.com/locate/mcm

Differential fault analysis on block cipher SEED☆

Kitae Jeong [a], Yuseop Lee [a], Jaechul Sung [b,*], Seokhie Hong [a]

[a] Center for Information Security Technologies (CIST), Korea University, Seoul, Republic of Korea

[b] Department of Mathematics, University of Seoul, Seoul, Republic of Korea

ARTICLE INFO

Article history:
Received 13 September 2010
Accepted 5 January 2011

Keywords:
Differential fault analysis
Block cipher
SEED

ABSTRACT

SEED is a Korean standard block cipher, and it is chosen as a 128-bit ISO/IEC standard block cipher together with AES and Camellia. In this paper, we propose a differential fault analysis on SEED on the basis of the bit-oriented model. Our fault model on SEED-128 is more flexible than the previous fault model on SEED-128. And our attack results on SEED-192/256 are the first known cryptanalytic results. From the simulation results, our attack on SEED-128 can recover a 128-bit secret key within a few seconds by inducing four faults. However, the computational complexities of our attack on SEED-192/256 are impractical.

1. Introduction

In general, the security of block ciphers has been measured by the attacks which use the theoretical weakness of target algorithms, such as differential cryptanalysis [1] and linear cryptanalysis [2], and the attacks which use the extra information obtained during cryptographic computation of cryptosystems, such as side channel attacks.

Side channel attacks exploit the easily accessible information like power consumption, running time and input–output behavior under malfunctions, and they can be mounted by anyone using low-cost equipment. They amplify and evaluate the leaked information with the help of statistical methods, and are often much more powerful than classical cryptanalysis.

Differential fault analysis (DFA), one of the side channel attacks, was first proposed by Biham and Shamir on DES in 1997 [3]. This attack exploits faults within the computation of a cryptographic algorithm to reveal the secret information. So far, DFAs on many block ciphers have been proposed [4–8]. In most DFAs, the attacker induces a random fault in some round of the encryption process and obtains a faulty ciphertext. By differential analysis, the last round key can be recovered. Then the attacker can decrypt the right ciphertext to obtain the input of the last round, which is the output of the penultimate round. And the attacker repeats the above procedure to recover more round keys until a secret key is obtained by the key schedule.

SEED is a 128-bit block cipher which supports 128-, 192- and 256-bit secret keys [9,10]. In this paper, we call this algorithm SEED-128/192/256, respectively. SEED-128/192/256 have the Feistel structure with 16/20/24 iterative rounds. SEED-128 was adopted as a national industrial association standard (TTAS KO-12.0004) in 1999, and ISO/IEC 18033-3 [11] and IETF RFC 4269 adopted it in 2005. SEED-192/256 were proposed in 2009 in order to enhance the applicability to various environments such as wire/wireless communications and electronic commerce. These algorithms have been adopted by most security systems in Korea. They are designed to utilize S-boxes and permutations that balance with the current computing technology. It is known that these algorithms have sufficient security against differential cryptanalysis and linear cryptanalysis. The theoretical attack result is only a differential cryptanalysis on a 7-round reduced SEED-128 [12].

☆ This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0000261).

* Corresponding author.
E-mail addresses: kite@cist.korea.ac.kr (K. Jeong), yusubi@cist.korea.ac.kr (Y. Lee), jcsung@uos.ac.kr (J. Sung), hsh@cist.korea.ac.kr (S. Hong).

0895-7177/$ – see front matter
doi:10.1016/j.mcm.2011.01.008

In 2004, Yoo et al. proposed a DFA on SEED-128 [13]. In this attack, it is assumed that the attacker induces permanent faults on the whole left register of round 15. Thus the attacker can obtain a secret key by using only two fault ciphertexts for the encryption and decryption processes, respectively. In this paper, we propose DFAs on SEED-128/192/256. The attack results on SEED-192/256 are the first known cryptanalytic results. In our proposed fault models, 1-bit random faults are induced to input registers of the last G function in the target round. So our fault models are more flexible than that of [13].

We propose two versions of DFA on SEED-128: a basic attack and an improved attack. The simulation results of the basic attack show that we can recover a 128-bit secret key within a few seconds with 48 faults and the computational complexity of about 2^{33} arithmetic operations. However, this attack requires many faults and operations. So we solve these problems by using three techniques in the improved attack. From the simulation results, our attack can recover a 128-bit secret key within a few seconds by using only four faults. In the case of SEED-192/256, we cannot apply the third technique of the improved attack, because of the complicated key schedules. So the computational complexities of these attacks are impractical, about 2^{65} and 2^{97} arithmetic operations, respectively. However, our attacks consist of simple arithmetic operations and need a small number of faults. So, these attacks can be meaningful if they are executed in more efficient platforms.

This paper is organized as follows. In Section 2, we briefly introduce SEED-128/192/256. In Section 3, we present our fault assumptions and the basic idea of our attacks. The basic attack and the improved attack on SEED-128 are described in Sections 4 and 5, respectively. In Section 6, we present the DFA on SEED-192/256. Finally, we give our conclusion in Section 7.

2. Description of SEED

SEED-128/192/256 is a 128-bit block cipher which supports 128-, 192- and 256-bit secret keys, respectively. Throughout this paper, the following notations are used.

- &: bitwise AND.
- $\boxplus(\boxminus)$: addition (subtraction) in modular 2^{32}.
- $\lll$ $(\ggg)n$: left (right) circular rotation by n bits.
- $\|$: concatenation.

The structure of SEED-128/192/256 is shown in Fig. 1. Here, the number of rounds Nr is 16 for SEED-128, 20 for SEED-192 and 24 for SEED-256. A 128-bit plaintext is divided into two 64-bit sub-blocks $(L_0, R_0) = (L_{0,0}\|L_{0,1}, R_{0,0}\|R_{0,1})$, and a 128-bit ciphertext $(L_{Nr}, R_{Nr}) = (L_{Nr,0}\|L_{Nr,1}, R_{Nr,0}\|R_{Nr,1})$ is generated by iterating the round function FNr times.

The round function F has the 3-round modified MISTY structure (see Fig. 2(a)) and consists of a G function. In round i, it takes a 64-bit value $R_{i-1} = R_{i-1,0}\|R_{i-1,1}$ and a 64-bit round key $RK_i = RK_{i,0}\|RK_{i,1}$ as input values. As shown in Fig. 2(b), the G function has the following two layers. See [10] for detailed descriptions of the G function.

- SL layer: two 8×8 S-boxes, S_1 and S_2.
- DL layer: a block permutation of sixteen 8-bit sub-blocks $(G(X) = DL(SL(X)))$.

Fig. 3 presents key schedules of SEED-128/192/256. They use the G function, addition, subtraction and left/right circular rotation, and generate 64-bit round keys $(RK_{i,0}, RK_{i,1})$ for a total 16/20/24 rounds by using 128-/192-/256-bit secret keys, respectively. Here, round constants KC_i are generated as follows $(i = 1, \ldots, 15)$:

$$KC_0 = \texttt{0x9e3779b9}, \qquad KC_i = (KC_0)_{\lll i}.$$

Rotation parameters 8 for SEED-128 and $rot(=9, 8, 12)$ for SEED-192/256 are used respectively to satisfy the on-the-fly encryption. For detailed descriptions of key schedules, see [9,10].

3. Fault assumptions and the basic idea

As shown in Fig. 2(a), the G function is iterated three times in the round function F. Generally, the design of this structure on hardware platforms is the iteration of the G function three times rather than the implementation of an all-in-one F. Thus it is likely that faults are induced to the input registers of the G function.

Our proposed fault model includes the following assumptions.

- The attacker has the capability to choose one plaintext to encrypt and obtain the corresponding right/faulty ciphertexts.
- The attacker can induce 1-bit random faults to input registers of the last G function in the target round.
- The location and value of faults are both unknown.

The main attack procedure on SEED-128 is as follows. The attacks on SEED-192/256 are similar to that on SEED-128. Given a 128-bit plaintext P, we denote 128-bit right/faulty ciphertexts as follows. Here, $C_L = (C_{L,0}, C_{L,1})$ and $C_R = (C_{R,0}, C_{R,1})$.

- (P, C): a right plaintext/ciphertext pair $(C = (C_L, C_R))$.
- (P, C^i): a faulty plaintext/ciphertext pair for an i-th fault $(C^i = (C_L^i, C_R^i))$.

First, by inducing 1-bit random faults to the input register of the third G function in the last round, we get the output value $X_{16,0}$ of the second G function and the input value $X_{16,1}$ of the third G function in the last round. Second, we obtain

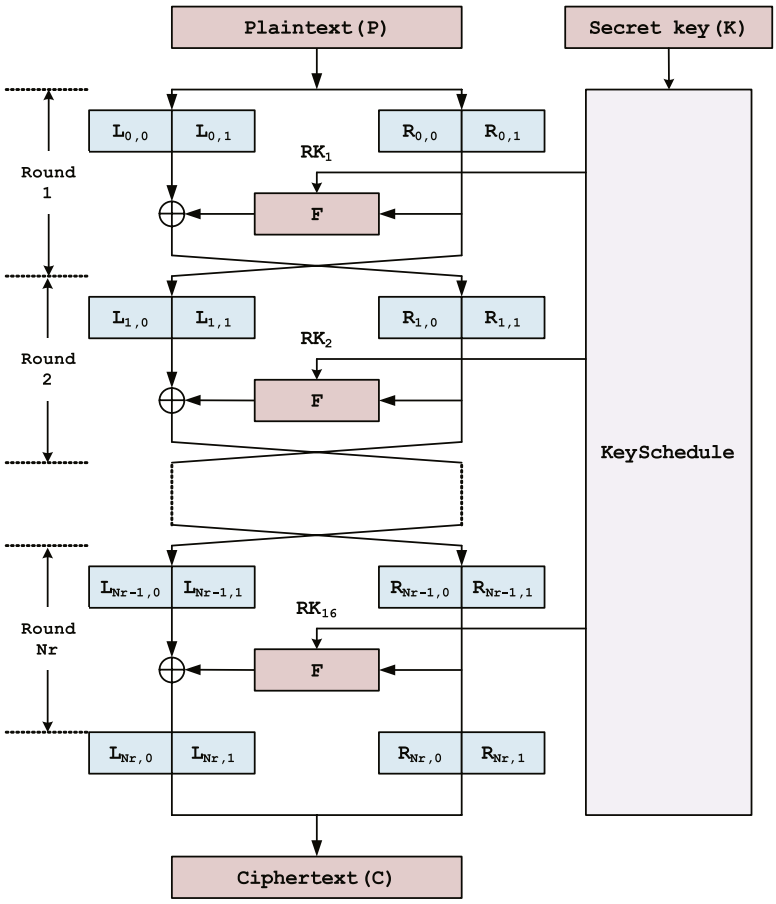

Fig. 1. The structure of SEED-128/192/256.

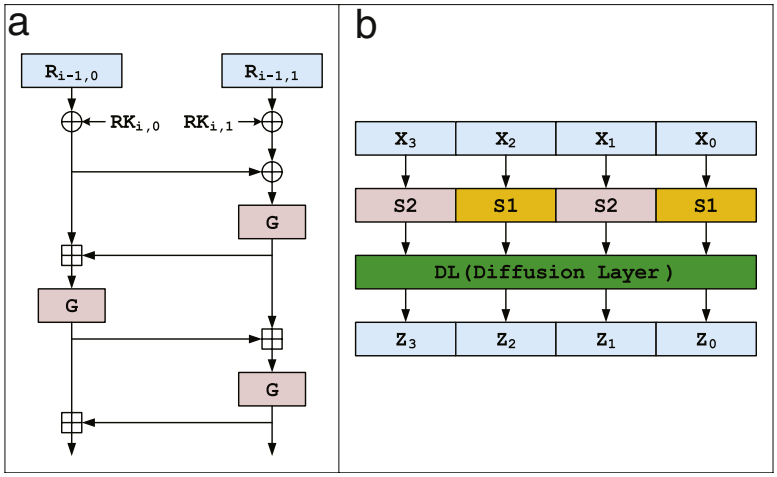

Fig. 2. (a) *F* function and (b) *G* function.

$X_{15,0}$ and $X_{15,1}$ by repeating the above procedure in round 15. And then, RK_{15} and RK_{16} are computed by using $(X_{15,0}, X_{15,1})$ and $(X_{16,0}, X_{16,1})$. Finally, we recover the secret key by using (RK_{15}, RK_{16}).

There are two versions of our DFA on SEED-128: the basic attack and the improved attack. In the basic attack, we obtain the exact intermediate value of particular registers. However, this attack uses many faults. Thus we solve this problem by using three techniques in the improved attack. In the case of SEED-192/256, we cannot apply the third technique of the improved attack to them. The reason is explained in Section 6.

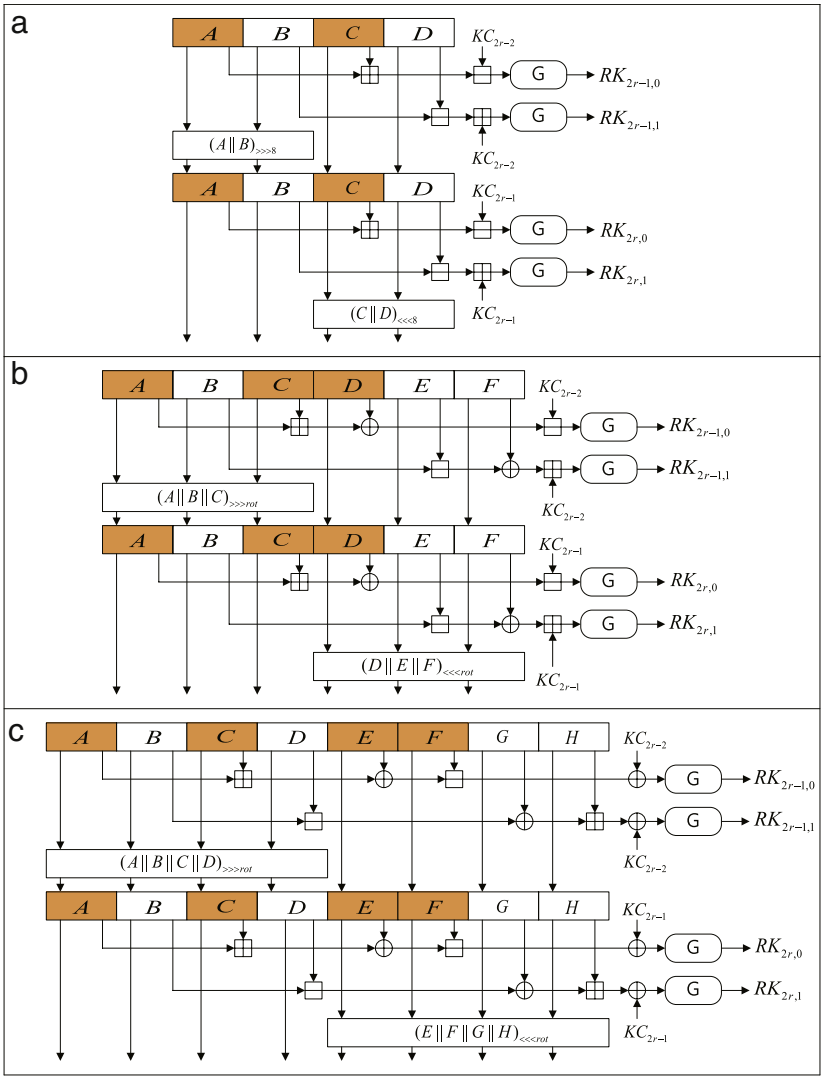

Fig. 3. Key schedules of (a) SEED-128, (b) SEED-192 and (c) SEED-256.

4. Basic DFA on SEED-128

In this section, we propose the basic attack on SEED-128. This attack consists of four parts: the computation of $(X_{16,0}, X_{16,1})$, $(X_{15,0}, X_{15,1})$, (RK_{15}, RK_{16}) and the recovery of a 128-bit secret key.

4.1. Computation of $(X_{16,0}, X_{16,1})$

We induce 1-bit random faults to the input register of the third G function in round 16. Fig. 4 represents our DFA model in round 16. Here, Δ^i means an i-th fault. That is, the input value $X^i_{16,1}$ of the third G function is $X_{16,1} \oplus \Delta^i$ in the case of a faulty plaintext/ciphertext pair.

This step consists of the following two substeps: we first obtain $X_{16,1}$ by using difference distribution tables of two S-boxes, and then $(X_{16,0}, X_{16,1})$ is computed by using the property of addition and XOR.

4.1.1. Computing $X_{16,1}$

The output difference γ^i of the third G function is computed using the following equation.

$$\gamma^i = (C_{L,1} \oplus L_{15,1}) \oplus (C^i_{L,1} \oplus L_{15,1}) = C_{L,1} \oplus C^i_{L,1}.$$

Thus we can compute the exact value of γ^i by using right/faulty ciphertexts, though we do not know $L_{15,1}$.

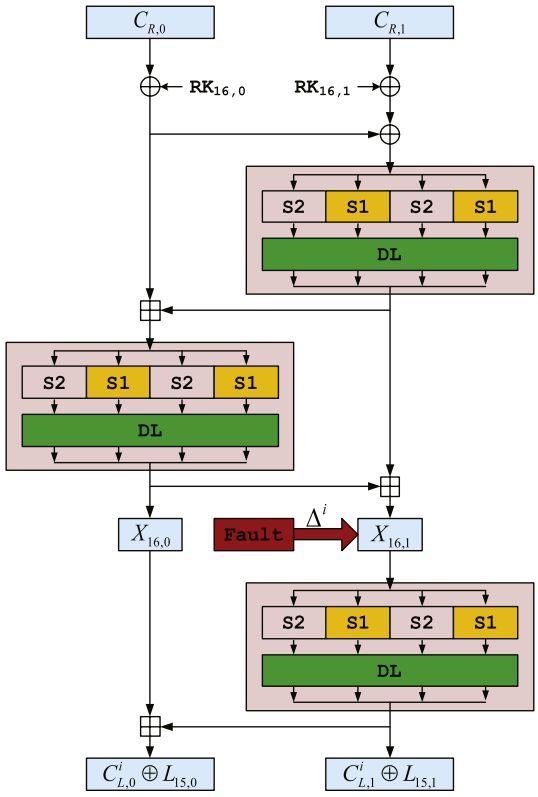

Fig. 4. DFA model in round 16.

On the other hand, since the *DL* layer used in the *G* function is a linear permutation ($G(\cdot) = DL(SL(\cdot))$), we can easily compute the output difference $\beta^i = (\beta^i_3, \beta^i_2, \beta^i_1, \beta^i_0)$ of the *SL* layer, which is equal to $DL^{-1}(\gamma^i)$. From our fault assumption that 1-bit random faults are induced, a 32-bit value β^i has one active byte difference and three nonactive byte differences. And the input difference $\alpha^i = (\alpha^i_3, \alpha^i_2, \alpha^i_1, \alpha^i_0)$ of the *SL* layer has the form 2^j $(0 \le j \le 7)$. Thus if we precompute difference distribution tables of two *S*-boxes, S_1 and S_2, as follows ($k = 0, 1, 2, 3$), we can obtain candidates of $X_{16,1}$.

$$S_1[\alpha^i_k][\beta^i_k] = \{x | S_1(x \oplus \alpha^i_k) \oplus S_1(x) = \beta^i_k\},$$
$$S_2[\alpha^i_k][\beta^i_k] = \{x | S_2(x \oplus \alpha^i_k) \oplus S_2(x) = \beta^i_k\}.$$

For each *S*-box, the number of possible α^i_k and β^i_k is 8 and 255, respectively. Given one (α^i_k, β^i_k), the number of x is one of 0, 2 and 4 from a difference distribution table of each *S*-box. Given a β^i_k, there exist 128, 126 and 1 α^i_k, which makes number of x 0, 2 and 4, respectively. Thus we obtain one 8-bit value of $X_{16,1}$ from one (α^i_k, β^i_k) on average. Since the number of possible α^i_k is 8, we get eight candidates of 8-bit value of $X_{16,1}$ on average. By repeating this procedure at other byte positions, we obtain some candidates of $X_{16,1}$. Until the number of candidates of $X_{16,1}$ is at most 2, we repeat this procedure by using more faults.

Since difference distribution tables are constructed in the precomputation phase, the computational complexity of this substep is negligible. We simulated this substep on a PC using Visual studio 2008 on Intel® Core™ i7 CPU 2.80 GHz with 6.00 GB memory. The fault induction was simulated by computer software. The simulation result shows that 24 faults need to obtain at most 2 candidates of $X_{16,1}$ with a probability of 0.95.

4.1.2. Computing $(X_{16,0}, X_{16,1})$

Given candidates of $X_{16,1}$, $X_{16,0}$ is recovered by using the property of addition and XOR. The left 32-bit output difference δ^i of round 16 is computed as follows (see Fig. 4).

$$\delta^i = (C_{L,0} \oplus L_{15,0}) \oplus (C^i_{L,0} \oplus L_{15,0}) = C_{L,0} \oplus C^i_{L,0}.$$

Recall that the output difference γ^i of the third *G* function is equal to $C_{L,1} \oplus C^i_{L,1}$. Thus, the right $(X_{16,0}, X_{16,1})$ should satisfy Eq. (1).

$$(X_{16,0} \boxplus G(X_{16,1})) \oplus (X_{16,0} \boxplus (G(X_{16,1}) \oplus \gamma^i)) = \delta^i. \tag{1}$$

Lipmaa and Moriai introduced the method of computing differential properties of addition in [14]. To compute Eq. (1), we apply the results of [14] to this equation. From the notations of [14], the probability that Eq. (1) holds is $DP^{+}((0, \gamma^i) \to \delta^i)$. Given $G(X_{16,1})$, $G(X_{16,1}) \oplus \gamma^i$ and δ^i, we compute candidates of $X_{16,0}$. Since the number of possible $X_{16,0}$ is 2^{32}, we can get candidates of $(X_{16,0}, X_{16,1})$ by using simple arithmetic operations. Their number depends on $DP^{+}((0, \gamma^i) \to \delta^i)$.

The computational complexity of this substep is about 2^{32} arithmetic operations. We simulated this substep on a PC. From the simulation result, we can obtain two candidates of $(X_{16,0}, X_{16,1})$ by using 24 faults with a probability of 0.99.

4.2. Computation of $(X_{15,0}, X_{15,1})$

The attack procedure to obtain candidates of $(X_{15,0}, X_{15,1})$ is similar to that to compute candidates of $(X_{16,0}, X_{16,1})$. First, we induce 1-bit random faults to the input register of the third G function in round 15 and obtain the corresponding right/faulty ciphertexts for the same plaintext. And then we compute candidates of $(X_{15,0}, X_{15,1})$ by using the same procedure as that in the previous subsection.

This procedure does not use candidates of $(X_{16,0}, X_{16,1})$, since (C_L, C_L^i) used in the previous subsection is replaced with (C_R, C_R^i) (see Fig. 1). $(X_{16,0}, X_{16,1})$ is simply used to recover (RK_{15}, RK_{16}) together with $(X_{15,0}, X_{15,1})$. Thus we can implement this procedure and that to compute candidates of $(X_{16,0}, X_{16,1})$ in parallel. So the computational complexity of this step is not included in the total computational complexity. The simulation result of this step is the same as that in the previous subsection.

4.3. Computation of (RK_{15}, RK_{16})

For each candidate of $(X_{16,0}, X_{16,1})$, the round key RK_{16} of round 16 is computed by using the following equations (see Fig. 3(a)).

$$RK_{16,0} = C_{R,0} \oplus (G^{-1}(X_{16,0}) \boxminus (X_{16,1} \boxminus X_{16,0})),$$
$$RK_{16,1} = G^{-1}(X_{16,1} \boxminus X_{16,0}) \oplus C_{R,1} \oplus RK_{16,0}.$$

Here, $RK_{16} = (RK_{16,0}, RK_{16,1})$.

The method to compute candidates of RK_{15} is similar to that to compute candidates of RK_{16}. That is, for each candidate of $(X_{15,0}, X_{15,1})$, RK_{15} is computed by using the following equations.

$$RK_{15,0} = L_{15,0} \oplus (G^{-1}(X_{15,0}) \boxminus (X_{15,1} \boxminus X_{15,0})),$$
$$RK_{15,1} = G^{-1}\left(X_{15,1} \boxminus X_{15,0}\right) \oplus L_{15,1} \oplus RK_{15,0}.$$

Here, $(L_{15,0}, L_{15,1})$ is a left 32-bit input value of round 16, and it is computed by using candidates of RK_{16} (see Fig. 1).

Since this step uses only four simple arithmetic operations for each (RK_{15}, RK_{16}), the computational complexity of this procedure is negligible. From the simulation result, we get four candidates of (RK_{15}, RK_{16}) by using 48 faults with a probability of 0.99.

4.4. Recovery of a 128-bit secret key

The procedure to generate 16 round keys of SEED-128 is as follows. First, a 128-bit secret key $K = (K_3, K_2, K_1, K_0)$ is divided into four 32-bit registers (A, B, C, D). And then 16 round keys are generated, as depicted in Fig. 3(a). Since this algorithm satisfies the on-the-fly encryption, (A, B, C, D) is equal to (K_3, K_2, K_1, K_0) after generating all round keys. Thus, if we get (A, B, C, D) before generating the round key of round 16, we can easily compute K, which is equal to $A\|B\|(C\|D)_{\lll 8}$ (see Fig. 3(a)).

From Fig. 3(a), $RK_{15} = (RK_{15,0}, RK_{15,1})$ and $RK_{16} = (RK_{16,0}, RK_{16,1})$ are computed as follows. Here, KC_{14} and KC_{15} are 32-bit round constants and $A'\|B' = (A\|B)_{\lll 8}$.

$$\begin{aligned} RK_{15,0} &= G(A' \boxplus C \boxminus KC_{14}),\\ RK_{15,1} &= G(B' \boxminus D \boxplus KC_{14}),\\ RK_{16,0} &= G(A \boxplus C \boxminus KC_{15}),\\ RK_{16,1} &= G(B \boxminus D \boxplus KC_{15}). \end{aligned} \tag{2}$$

If x and y are candidates of C and D, respectively, then, for each x and y, candidates of A and B are computed as follows.

$$A(x) = G^{-1}(RK_{16,0}) \boxminus x \boxplus KC_{15},$$
$$B(y) = G^{-1}(RK_{16,1}) \boxplus y \boxminus KC_{15},$$
$$A'(x) = G^{-1}(RK_{15,0}) \boxminus x \boxplus KC_{14},$$
$$B'(y) = G^{-1}(RK_{15,1}) \boxplus y \boxminus KC_{14}.$$

For each x and y, $(A(x), B(y), A'(x), B'(y))$ should satisfy the following conditions from the property of key schedule. The 24 most significant bits of $A'(x)$ are equal to the 24 least significant bits of $A(x)$, and the 24 most significant bits of $B'(y)$ are

Table 1
Number of candidates for the number of injected faults.

Total number of injected faults	Number of candidates		
	$(X_{16,0}, X_{16,1})$	$(X_{15,0}, X_{15,1})$	(RK_{15}, RK_{16})
16	7.0	5.9	33.1
18	5.2	6.3	23.1
20	3.3	3.0	9.7
22	2.6	2.6	6.9

equal to the 24 least significant bits of $B(x)$. That is, $(A(x), B(y), A'(x), B'(y))$ should satisfy the following equations.

$$A(x)\&0\text{x}00\text{ffffff} = (A'(x)\&0\text{xffffff}00)_{\ggg 8},$$
$$B(y)\&0\text{x}00\text{ffffff} = (B'(y)\&0\text{xffffff}00)_{\ggg 8}.$$

From the above equations, we obtain 2^8 candidates of x and y, respectively. Moreover, the total of 2^{16} candidates of (x, y) should satisfy the following conditions. The 8 least significant bits of $A'(x)$ are equal to the 8 most significant bits of $B(y)$, and the 8 most significant bits of $A(x)$ are equal to the 8 least significant bits of $B'(y)$. That is, $(A(x), B(y), A'(x), B'(y))$ should satisfy the following equations.

$$A'(x)\&0\text{x}000000\text{ff} = (B(y)\&0\text{xff}000000)_{\ggg 24},$$
$$B'(y)\&0\text{x}000000\text{ff} = (A(x)\&0\text{xff}000000)_{\ggg 24}.$$

Applying this procedure, we obtain candidates of K.

We simulated this step on a PC. From the simulation result, the number of candidates of K is at most 2. Thus, by using one trial encryption with a right plaintext/ciphertext pair, we can recover a 128-bit secret key K by using about 2^{33} arithmetic operations. A simulation result shows that we can always recover the exact secret key within a few seconds, if we use a total of 48 faults.

In summary, the computational complexity of basic attack on SEED-128 is about 2^{33} arithmetic operations and the number of required faults is 48. And our attack algorithm can always recover a 128-bit secret key within a few seconds.

5. Improved DFA on SEED-128

As mentioned in the previous subsection, our basic DFA on SEED-128 requires many faults (48 1-bit random faults) and many operations (about 2^{33} arithmetic operations). To overcome these problems, we introduce three techniques in this section.

The first idea is to decrease the number of required faults. In the basic attack, we reduce the candidates of $X_{i,0}$ and $X_{i,1}$ as much as possible by using many faults ($i = 15, 16$). For example, their number is two in our simulation. In the improved attack, we increase the number of them. This technique results in bigger computational complexity and smaller number of required faults. The increased computational complexity is resolved in the second and third ideas.

We simulated the improved attack applying the first idea on a PC. We ran this attack 1000 times. Table 1 presents the average number of candidates of $(X_{16,0}, X_{16,1})$, $(X_{15,0}, X_{15,1})$ and (RK_{15}, RK_{16}). Note that the number of faults in round 15 is equal to that in round 16. For example, in the case that the total number of injected faults is 16, the number of faults in round 15 and round 16 is 8, respectively. From Table 1, the bigger the number of injected faults is, the smaller the number of candidates is.

The second idea is to decrease the computational complexity computing candidates of $X_{15,0}$ and $X_{16,0}$ from $X_{15,1}$ and $X_{16,1}$, respectively. We apply the results of [14] to our attack. In the basic attack, the computational complexity of about 2^{32} arithmetic operations is required to compute Eq. (1). Recall that the numbers of candidates of $(X_{15,0}, X_{15,1})$ and $(X_{16,0}, X_{16,1})$ depend on $DP^+((0, \gamma^i) \to \delta^i)$. Thus, for each candidate of $X_{15,1}$ and $X_{16,1}$, if $DP^+((0, \gamma^i) \to \delta^i)$ is equal to zero, we discard it. Only the surviving candidates, that is those for which this probability is not equal to zero, are used. So, we can reduce the computational complexity.

The last idea is to decrease the computational complexity of computing candidates of K. The basic attack needs about 2^{33} arithmetic operations to compute candidates of K from candidates of (RK_{15}, RK_{16}). Thus, if the number of candidates of (RK_{15}, RK_{16}) increases, the computational complexity rapidly increases. In the improved attack, we carry out a byte-oriented computation. In detail, from Eq. (2), the 32-bit registers (A, B, C, D) satisfy the following equations.

$$\begin{aligned} A \boxplus C &= G^{-1}(RK_{16,0}) \boxplus KC_{15}, \\ A' \boxplus C &= G^{-1}(RK_{15,0}) \boxplus KC_{14}, \\ B \boxminus D &= G^{-1}(RK_{16,1}) \boxminus KC_{15}, \\ B' \boxminus D &= G^{-1}(RK_{15,1}) \boxminus KC_{14}. \end{aligned} \tag{3}$$

Here, $A = (a_3, a_2, a_1, a_0), B = (b_3, b_2, b_1, b_0), A' = (b_0, a_3, a_2, a_1)$ and $B' = (a_0, b_3, b_2, b_1)$.

Table 2
Results of the improved attack on SEED-128.

Total number of injected faults	Required time (s)			Success probability (%)
	Best	Worst	Average	
12	0.2	16134.2	125.2	100
14	0.2	5891.1	63.0	100
16	0.2	641.6	4.3	100
18	0.2	929.7	3.4	100
20	0.2	178.7	0.8	100
22	0.2	18.1	0.38	100

Since we know the values of the right-hand side of Eq. (3), we can compute $A \boxminus A'$ and $B \boxminus B'$ from them. Here, we denote these values by $W = (w_3, w_2, w_1, w_0)$ and $Z = (z_3, z_2, z_1, z_0)$, respectively. That is, the following equations hold.

$$(w_3, w_2, w_1, w_0) = (a_3, a_2, a_1, a_0) \boxminus (b_0, a_3, a_2, a_1),$$
$$(z_3, z_2, z_1, z_0) = (b_3, b_2, b_1, b_0) \boxminus (a_0, b_3, b_2, b_1).$$

First, for each candidate of (RK_{15}, RK_{16}), we guess a_0 and compute a_1 from $w_0 = a_0 - a_1$. And then we compute a_2 from $(w_1, w_0) = (a_1, a_0) - (a_2, a_1)$. Repeating this procedure, we compute a_3 and b_0. Similarly, from Z, we compute b_1, b_2, b_3 and a_0. Thus, we check that the guessed a_0 is equal to the computed a_0. If it is, (C, D) is computed by using Eq. (3). The probability that a wrongly guessed a_0 passes this procedure is 2^{-8}. So, one candidate of (RK_{15}, RK_{16}) results in one K on average. Since the computational complexity of this procedure is about 8×2^8 arithmetic operations, we can reduce the computational complexity.

We simulated our improved attack on a PC 1000 times. Our results are shown in Table 2. For each number of injected faults, our attack can always recover a 128-bit secret key, and takes 0.2 s in the best case. However, the gap between the best required time and the worst required time increases when the number of injected faults decreases. The reason for this problem is the use of random faults. In detail, we assume that all random faults are injected in the same byte position. Then only one byte position of $X_{16,1}$ (or $X_{15,1}$) has a small number of candidates and other byte positions have many candidates, 2^8. So our attack has many candidates and the computational complexity increases rapidly, too. This phenomenon happens more frequently in the case of few faults rather than in that of many faults. In the base case where our attack takes 0.2 s to recover a secret key, faults are injected to all four byte positions.

We test the case that the number of faults is small. When the number of faults is eight, the best execution time is a few seconds. However, the possibility of this case is very low. Most cases are done within one hour. The worst execution time is about 20 hours. In this case, faults are induced to two byte positions in round 15 and 16, respectively. So our attack is executable by using four faults or fewer, if we consider the trade-offs between the number of faults and the execution time. On the other hand, we can check the exact byte positions of the injected faults by examining the difference between the ciphertexts, $C_{L,1} \oplus C^i_{L,1}$ (see Fig. 1). This means that our attack can always recover a 128-bit secret key of SEED-128 within a few seconds by using only faults which are injected to the required byte positions.

6. DFA on SEED-192/256

In this section, we propose DFAs on SEED-192/256. In these attacks, 1-bit random faults are injected to the same position as the attack on SEED-128 in the last three rounds and the last four rounds, respectively.

The attack procedure on SEED-192 is summarized as follows. The attack procedure on SEED-256 is similar to that on SEED-192.

1. [**Collection of right ciphertext**] Choose a plaintext P and obtain the corresponding right ciphertext $C = (C_L, C_R)$.
2. [**Collection of faulty ciphertext**] After inducing an i-th 1-bit random fault Δ^i to the input register of the third G function in the last round, obtain the corresponding faulty ciphertext $C^i = (C^i_L, C^i_R)$ $(i = 1, \ldots, n)$.
3. [**Computation of candidates of** RK_{20}] Compute candidates of RK_{20} by using the same method as that to get candidates of RK_{16} in SEED-128.
4. [**Collection of faulty ciphertext**] After inducing an i-th 1-bit random fault Δ^i to the input register of the third G function in round 18 and 19, obtain the corresponding faulty ciphertext $C^i = (C^i_L, C^i_R)$ $(i = 1, \ldots, n)$.
5. [**Computation of candidates of** $(RK_{18}, RK_{19}, RK_{20})$] For each candidate of RK_{20}, compute the output value of round 19 and then obtain candidates of $(RK_{18}, RK_{19}, RK_{20})$ by using the same method as that to get candidates of (RK_{15}, RK_{16}) in SEED-128.
6. [**Recovery of a 192-bit secret key**] For each candidate of $(RK_{18}, RK_{19}, RK_{20})$, compute candidates of a 192-bit secret key by using the similar technique as that to obtain candidates of K in SEED-128 and then recover a 192-bit secret key by using one trial encryption.

Unfortunately, we cannot apply the third idea of the improved attack on SEED-128 to SEED-192/256. The reason is that the key schedules of SEED-192/256 are more complicated than that of SEED-128. Since an additional operation XOR and different circular rotation parameters are used in the key schedules of SEED-192/256, it is difficult to use the third idea of

improved attack on SEED-128. Thus, we apply only techniques of the basic attack on SEED-128 to these attacks. In detail, we guess two 64-bit values (A, C) and (B, E) in the attack on SEED-192 and two 96-bit values (A, C, E) and (B, D, G) in SEED-256.

So the computational complexities of these attacks are about 2^{65} and 2^{97} arithmetic operations, respectively. These values are impractical. However, our attacks consist of simple arithmetic operations and need a small number of faults. So, these attacks can be meaningful if they are executed in more efficient platforms.

7. Conclusion

In this paper, we have presented DFAs on SEED. Our attack on SEED-128 is executed within a few seconds by using only four faults. And our attacks on SEED-192/256 need a small number of faults, though the computational complexities of them are impractical. We believe that it is a good topic to study the method of reducing the computational complexities of our DFAs on SEED-192/256.

References

[1] E. Biham, A. Shamir, Differential cryptanalysis of DES-like cryptosystem, Journal of Cryptology 4 (1) (1991) 3–72.
[2] M. Matsui, Linear cryptanalysis method for DES cipher, in: Eurocrypt 1993, in: LNCS, vol. 765, Springer-Verlag, 1994, pp. 386–397.
[3] E. Biham, A. Shamir, Differential fault analysis of secret key cryptosystems, in: Crypto 1997, in: LNCS, vol. 1294, Springer-Verlag, 1997, pp. 513–525.
[4] H. Chen, W. Wu, D. Feng, Differential fault analysis on CLEFIA, in: ICICS 2007, in: LNCS, vol. 4861, Springer-Verlag, 2007, pp. 284–295.
[5] C. Clavier, B. Gierlichs, I. Verbauwhede, Fault analysis study of IDEA, in: CT-RSA 2008, in: LNCS, vol. 4964, Springer-Verlag, 2008, pp. 247–287.
[6] L. Hemme, A differential fault analysis against early rounds of (Triple-)DES, in: CHES 2004, in: LNCS, vol. 3156, Springer-Verlag, 2006, pp. 254–267.
[7] W. Li, D. Gu, J. Li, Differential Fault Analysis on the ARIA Algorithm, in: Information Sciences, vol. 178, Elsevier, 2008, pp. 3727–3737.
[8] A. Moradi, M. Shalmani, M. Salmasizadeh, A generalized method of differential fault attack against AES cryptosystem, in: CHES 2006, in: LNCS, vol. 4249, Springer-Verlag, 2006, pp. 91–100.
[9] K. Jeong, J. Choi, Y. Lee, C. Lee, J. Sung, H. Park, Y. Kang, Update on SEED: SEED-192/256, in: ISA 2009, in: LNCS, vol. 5576, Springer-Verlag, 2009, pp. 1–10.
[10] Korea Internet & Security Agency, SEED 128 Algorithm Specification, Available at: http://seed.kisa.or.kr/seed/jsp/seed_1010.jsp.
[11] ISO/IEC 18033-3, Information Technology – Security Techniques – Encryption Algorithms – Part 3: Block Ciphers, ISO, 2005. Available at: http://www.iso.org/iso/catalogue_detail.htm?csnumber=37972.
[12] H. Yanami, T. Shimoyama, Differential cryptanalysis of a reduced-round SEED, in: SCN 2002, in: LNCS, vol. 2576, Springer-Verlag, 2003, pp. 186–198.
[13] H. Yoo, C. Kim, J. Ha, S. Moon, I. Park, Side channel cryptanalysis on SEED, in: WISA 2004, in: LNCS, vol. 3325, Springer-Verlag, 2004, pp. 411–424.
[14] H. Lipmaa, S. Moriai, Efficient algorithms for computing differential properties of addition, in: FSE 2001, in: LNCS, vol. 2355, Springer-Verlag, 2002, pp. 336–350.