Improved Fault Attack on FOX

Jianxiong You[1], Ruilin Li[1], Bing Sun[1], and Chao Li[1,2]

[1] Department of Mathematics and System Science, Science College, National University of Defense Technology, Changsha, 410073, China
jianxiongyou@gmail.com
[2]State Key Laboratory of Information Security, Institute of Software, Chinese Academy of Sciences, Beijing, 100190, China

Abstract. In this paper, we present an improved fault attack on the block cipher FOX64. Our improved method can deduce any round subkey through 4.25 faults on average (4 in the best case), and retrieve the whole round sub-keys through 45.45 faults on average (38 in the best case). Furthermore, it could be applied to other series of FOX.

Keywords: FOX, block cipher, Lai-Massay scheme, fault attack

1 Introduction

FOX [7], also known as IDEA-NXT, is a family of block ciphers designed by Junod and Vaudenay. The block size of FOX is either 64-bit or 128-bit, both of which have a variable key length ranging from 8 to 256 bits. The high-level structure of FOX is the so-called (extended) Lai-Massey scheme, and the round function is SPS style with three layers of round key addition.

Differential Fault Analysis (DFA) [1] attack was introduced by Biham and Shamir to DES-like secret key cryptosystems. Since then, it has been used to attack many other block ciphers, especially on AES, see e.g. [3-6,8-12].

In FDTC 2006, Breveglieri, Koren, and Maistri introduced a DFA attack on FOX [2]. Take FOX64 as an example, they showed that one round subkey could be retrieved by 11.45 faults on average using the random byte fault model, and the whole round sub-keys could be recovered by 183 faults on average. In this paper, we improve the fault attack on FOX64. Using our technique, the 64-bit subkey could be revealed by 4.25 faults on average, and the whole round sub-keys can be deduced through 45.45 faults on average.

The outline of this paper is as follows: we begin with a brief description of FOX in Section 2, and then present some properties related to the improved fault attack in Section 3. The proposed fault attack is introduced in Section 4. Section 5 demonstrates experimental results and Section 6 is the conclusion.

2 Description of FOX

We only introduce FOX64, for other series, see [7].

2.1 Encryption of FOX64

FOX64 has a 64-bit block size and a 128-bit key length. It iterates 15-times the round transformation **lmor64**, as illustrated in Fig.1, followed by a final round transformation called **lmid64**.

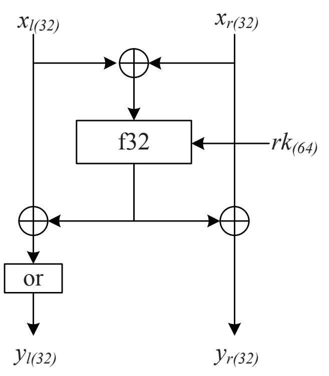

Fig. 1. Round transformation **lmor64**

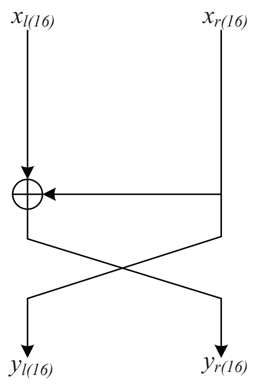

Fig. 2. Orthomorphism **or**

The round transformation **lmor64** transforms a 64-bit input $x_{(64)}$ and a 64-bit round key $rk_{(64)}$ into a 64-bit output $y_{(64)}$, which is defined as

$$\begin{aligned} y_{(64)} &= \texttt{lmor64}(x_{l(32)} \| x_{r(32)}, rk_{(64)}) \\ &= \texttt{or}(x_{l(32)} \oplus \texttt{f32}(x_{l(32)} \oplus x_{r(32)}, rk_{(64)})) \| (x_{r(32)} \oplus \texttt{f32}(x_{l(32)} \oplus x_{r(32)}, rk_{(64)})), \end{aligned}$$

where `f32` is the round function, and **or** is an orthomorphism.

The orthomorphism **or** is a function that takes a 32-bit input $x_{(32)}$ and returns a 32-bit output $y_{(32)}$, illustrated in Fig.2, which is defined by a simple Feistel transformation as

$$y_{l(32)} || y_{r(32)} = \textbf{or}(x_{l(16)} || x_{r(16)}) = x_{r(16)} || (x_{l(16)} \oplus x_{r(16)}).$$

The **lmid64** function is a slightly modified version of **lmor64**, where the transformation **or** is replaced by a identify transformation.

The encryption $c_{(64)}$ by FOX64 of a 64-bit plaintext $p_{(64)}$ is defined as

$$c_{(64)} = \texttt{lmid64}(\texttt{lmor64}(\cdots \texttt{lmor64}(p_{(64)}, rk_0), \cdots, rk_{14}), rk_{15}),$$

where rk_i, $i = 0, 1, \cdots, 15$, are round sub-keys generated through the key schedule from the master key.

2.2 Round Function f32

The round function `f32` consists of three main parts: a substitution part, denoted sigma4, a diffusion part, denoted mu4, and a round key addition part. Formally, the round function `f32` takes a 32-bit input $x_{(32)}$, a 64-bit round key $rk_{(64)} = rk_{0(32)}||rk_{1(32)}$ and returns

$$\begin{aligned} y_{(32)} &= \texttt{f32}(x_{(32)}, rk_{(64)}) \\ &= \texttt{sigma4}(\texttt{mu4}(\texttt{sigma4}(x_{(32)} \oplus rk_{0(32)}) \oplus rk_{1(32)})) \oplus rk_{0(32)}. \end{aligned}$$

The fuction `sigma4` takes a 32-bit input $x_{(32)} = x_{0(8)}\|x_{1(8)}\|x_{2(8)}\|x_{3(8)}$ and returns a 32-bit output $y_{(32)}$, it consists of 4 parallel computations of a non-linear mapping `sbox`, i.e.

$$\begin{aligned} y_{(32)} &= \texttt{sigma4}\left(x_{0(8)}\|x_{1(8)}\|x_{2(8)}\|x_{3(8)}\right) \\ &= \texttt{sbox}(x_{0(8)})\|\texttt{sbox}(x_{1(8)})\|\texttt{sbox}(x_{2(8)})\|\texttt{sbox}(x_{3(8)}). \end{aligned}$$

The function mu4 takes a 32-bit input $x_{0(8)}\|x_{1(8)}\|x_{2(8)}\|x_{3(8)}$ and returns a 32-bit output $y_{0(8)}\|y_{1(8)}\|y_{2(8)}\|y_{3(8)}$. It is defined by

$$\begin{pmatrix} y_{0(8)} \\ y_{1(8)} \\ y_{2(8)} \\ y_{3(8)} \end{pmatrix} = \begin{pmatrix} 1\ 1\ 1\ \theta \\ 1\ z\ \theta\ 1 \\ z\ \theta\ 1\ 1 \\ \theta\ 1\ z\ 1 \end{pmatrix} \times \begin{pmatrix} x_{0(8)} \\ x_{1(8)} \\ x_{2(8)} \\ x_{3(8)} \end{pmatrix},$$

where $\theta \in$GF(2^8) is the root of the irreducible polynomial $m(x) = x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + 1 \in \mathrm{GF}(2)[x]$ and $z = \theta^{-1} + 1$.

2.3 Key Schedule

The key schedule procedure of FOX64 generates 16 round sub-keys rk_i, $i = 0, 1, \ldots, 15$, from a master key K. Each round subkey is 64-bit, denoted as a concatenation of two 32-bit strings, i.e. $rk_i = rk_{i,0}\|rk_{i,1}$.

The key schedules of FOX series are very complex compared with other existing block ciphers, each subkey is related to the secret key and it is very difficult to acquire information about secret key or other subkeys from some certain subkeys. Due to this, in this paper, we assume that all subkeys are independent with each other. One can refer [7] for the detail of the key schedule.

3 Some Properties of FOX64

3.1 Property of Two-round Lai-Massay Scheme In a Fault Model

Consider a two-round Lai-Massay scheme in a fault model as shown in Fig.3. Let L_i and R_i be the left and right halves of the round input or output, where $i = 0, 1, 2$. Let A_j and D_j be the input and output of the bijective round function `f32`, where $j = 0, 1$. Assume a fault is induced into A_0, and denote the difference of a 32-bit state X as ΔX, then we have the following proposition:

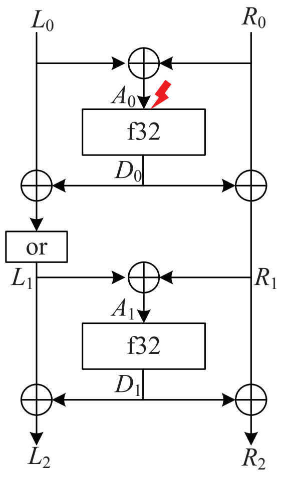

Fig. 3. Two round Lai-Massay scheme

Proposition 1. *Given a two-round Lai-Massay scheme as described above, assume $\Delta L_0 = 0$, $\Delta R_0 = 0$, and a fault is induced into A_0, i.e. $\Delta A_0 \neq 0$. Let $\Delta L_2 = (\alpha_0, \alpha_1, \alpha_2, \alpha_3)$, $\Delta R_2 = (\beta_0, \beta_1, \beta_2, \beta_3)$ be known values, then both the input difference and output difference of* `f32` *in the second round, i.e. ΔA_1 and ΔD_1, could be calculated.*

Proof. From $\Delta A_0 \neq 0$, we have $\Delta D_0 \neq 0$. Assume $\Delta D_0 = (x_0, x_1, x_2, x_3)$, then

$$\begin{cases} \Delta L_1 = \mathrm{or}(\Delta L_0 \oplus \Delta D_0) = \mathrm{or}(x_0, x_1, x_2, x_3) = (x_2, x_3, x_0 \oplus x_2, x_1 \oplus x_3) \\ \Delta R_1 = (x_0, x_1, x_2, x_3) \end{cases}$$

Notice that $\Delta A_1 = \Delta L_1 \oplus \Delta R_1 = \Delta L_2 \oplus \Delta R_2$, thus

$$\Delta A_1 = (\alpha_0 \oplus \beta_0, \alpha_1 \oplus \beta_1, \alpha_2 \oplus \beta_2, \alpha_3 \oplus \beta_3)$$

are known. Meanwhile, from

$$(x_2, x_3, x_0 \oplus x_2, x_1 \oplus x_3) \oplus (x_0, x_1, x_2, x_3) = (\alpha_0, \alpha_1, \alpha_2, \alpha_3) \oplus (\beta_0, \beta_1, \beta_2, \beta_3),$$

we get

$$(x_0, x_1, x_2, x_3) = (\alpha_2 \oplus \beta_2, \alpha_3 \oplus \beta_3, \alpha_0 \oplus \alpha_2 \oplus \beta_0 \oplus \beta_2, \alpha_1 \oplus \alpha_3 \oplus \beta_1 \oplus \beta_3).$$

Thus

$$\begin{aligned} \Delta D_1 = \Delta R_2 \oplus \Delta R_1 &= (\beta_0, \beta_1, \beta_2, \beta_3) \oplus (x_0, x_1, x_2, x_3) \\ &= (\alpha_2 \oplus \beta_0 \oplus \beta_2, \alpha_3 \oplus \beta_1 \oplus \beta_3, \alpha_0 \oplus \alpha_2 \oplus \beta_0, \alpha_1 \oplus \alpha_3 \oplus \beta_1). \end{aligned}$$

□

Remark 1. Proposition 1 also holds in the situation, where the second round transformation in the two-round Lai-Massay scheme contains the orthomorphism **or**. This is due to the simplicity of **or**, leading to easy calculation of the input from the output.

3.2 Property of the Substitution Layer sigma4

Given an 8×8 Sbox $S(\cdot)$, $\alpha, \beta \in \{0,1\}^8$, define $N_S(\alpha, \beta) = \#\{x \in \{0,1\}^8 : S(x) \oplus S(x \oplus \alpha) = \beta\}$. The differential property of an Sbox can be depicted by all the possible triplets $(\alpha, \beta, N_S(\alpha, \beta))$. The differential property of the Sbox employed in the substitution layer of FOX64 is summarized in table 1.

Table 1. Differential property of the Sbox

$N_S(\alpha, \beta)$	Frequency	$N_S(\alpha, \beta)$	Frequency
0	42871	10	19
2	15377	12	6
4	5758	16	70
6	680	256	1
8	754	–	–

Remark 2. Assume $S(\cdot)$ is the Sbox of FOX64, if $N_S(\alpha, \beta) \neq 0$, then the expected value of $N_S(\alpha, \beta) = 65536/(65536 - 42871) \approx 2.89$. This indicates that on average, one pair (α, β) could provide about 2.89 inputs x such that $S(x) \oplus S(x \oplus \alpha) = \beta$.

3.3 Property of the Diffusion Layer mu4

The differential branch number of `mu4` is 5, which implies that any input with one non-zero byte will lead to some output with four non-zero bytes. Moreover, the inversion of `mu4`, denoted as `mu4`$^{-1}$, can be expressed as follow:

$$\begin{pmatrix} a & c & d & e \\ a & d & e & c \\ a & e & c & d \\ b & a & a & a \end{pmatrix},$$

where $a = \theta^6 + \theta^5 + \theta^4 + \theta^3 + \theta^2 + \theta$, $b = \theta^7 + \theta^6 + \theta + 1$, $c = \theta^7 + \theta^6 + \theta^5 + 1$, $d = \theta^7 + \theta^5 + \theta^3 + \theta^2 + 1$ and $e = \theta^7 + \theta^5 + \theta^4$.

4 Improved Fault Attack On FOX64

4.1 Notations

- Assume all the round sub-keys, generated from the secret master key K through the key schedule, are $rk_i = rk_{i,0} \| rk_{i,1}$, $i = 0, 1, 2, \ldots, 15$.

- Denote a plaintext by $p = p_l \| p_r$, and the corresponding ciphertext by $c = E_K(p) = c_l \| c_r$. This ciphertext is called the *right* ciphertext and any indeterminate states corresponding to it are called the *right* indeterminate states.
- Consider the last round of FOX64: Let A_{16} denote the input of the round function `f32`; I_1 and B_{16} denote the input and output of the first substitution layer, respectively, i.e. $I_1 = A_{16} \oplus rk_{15,0}$, $B_{16} = \texttt{sigma4}(I_1)$; C_{16} denote the output of the diffusion layer, i.e. $C_{16} = \texttt{mu4}(B_{16})$; I_2 and D_{16} denote the input and output of the second substitution layer, i.e. $I_2 = C_{16} \oplus rk_{15,1}$, $D_{16} = \texttt{sigma4}(I_2)$.
- Given a 32-bit right state X, X^* denotes the *faulty* counterpart , and $\Delta X = X \oplus X^*$ denotes their difference.

4.2 Previous Fault Attack

As discussed in Section 2.3, due to the complexity of the key schedule, any fault attack on FOX64 aims to deduce all the round sub-keys.

Take the last round of FOX64 as an example, see Fig.4, both the previous fault attack and our improved fault attack (as described later) try to retrieve the 64-bit subkey by recovering the right intermediate state I_1 and I_2. Once I_1 and I_2 are known, one can do as follows:

- According to $I_1 = A_{16} \oplus rk_{15,0} = c_l \oplus c_r \oplus rk_{15,0}$, we thus have $rk_{15,0} = c_l \oplus c_r \oplus I_1$.
- According to $I_2 = C_{16} \oplus rk_{15,1} = \texttt{mu4}(B_{16}) \oplus rk_{15,1} = \texttt{mu4}(\texttt{sigma4}(I_1)) \oplus rk_{15,1}$, we thus have $rk_{15,1} = I_2 \oplus \texttt{mu4}(\texttt{sigma4}(I_1))$.

In order to recover the right state I_1 and I_2, previous fault attack adopts the random byte fault model and divides the attack procedure into the following two phases.

- In the first phase, the adversary injects faults into the calculation of round function, and the location is between the input of the round function and input of the diffusion layer. By using both the correct ciphertext and faulty ciphertexts, he applies differential cryptanalysis on the second substitution layer to recover I_2. The number of faults in this phase is about $2 \sim 8$ and 2.94 on average.
- In the second phase, the adversary also injects faults into the calculation of the round function and the location this time is only before the first substitution layer. He then applies differential cryptanalysis to the first substitution layer to recover I_1 based on the correct ciphertext and those faulty ciphertexts. The number of faults in this phase is about $8 \sim 28$ and 8.51 on average.

In total, about $8 \sim 31$ faults (11.45 on average) are needed to recover the right state I_1 and I_2.

4.3 General Idea of the Improved Fault Attack

We adopt the same attack model as in [2] and improve the efficiency of the previous fault attacks in the following two ways:

- In order to retrieve the 64-bit subkey for some certain round, say the i-th round, previous fault attack injects sufficient faults (about $8 \sim 31$) to the same i-th round, while ours is to induce faults at both the i-th and $(i-1)$-th round to deduce the subkey, thus it decreases the number of needed faults.
- In our improved attack, the faulty ciphertexts that are obtained when faults are injected in the i-th round are used twice (one for deducing the $(i+1)$-th round subkey, and the other for deducing the i-th roun subkey), thus it also decreases the number of needed faults (previous fault attack use those ciphertexts only once for deducing the i-th round subkey).

We briefly summarize the improved fault attack below:

1. Choose an arbitrary plaintext, encrypt it with the secret key and obtain the ciphertext. For the same plaintext, induce several random byte faults into the input of the round function in each round, obtain these faulty ciphertexts.
2. Deduce the last round subkey through the right ciphertext and the faulty ciphertexts that are obtained when faults are induced in the last round and penultimate round.
   (a) Consider the right ciphertext and faulty ciphertexts that are obtained when faults are induced into the last round, recover the right input state before the second substitution layer sigma4 in the last round.
   (b) Consider the right ciphertext and faulty ciphertexts that are obtained when faults are induced into the penultimate round, recover the right input state before the first substitution layer sigma4 in the last round.
   (c) Use the right intermediate states obtained from (a)(b) and the correct ciphertext, directly deduce the 64-bit subkey in the last round.
3. Recover sub-keys from the 2-dn round to the 15-th round in reverse order by using the same technique as in Step 2.
4. Recover the first round sub-key as described in Section 4.2.

4.4 Attack Procedure

Step 1 Data gathering. Choose an arbitrary plaintext $p = p_l \| p_r$, and obtain the right ciphertext $c = c_l \| c_r$ under the secret user key K. For the same plaintext p, induce several random byte faults into the input of the round function f32 in each round, and obtain these faulty ciphertexts.

Step 2 Recover the last round subkey $rk_{15} = rk_{15,0} \| rk_{15,1}$.

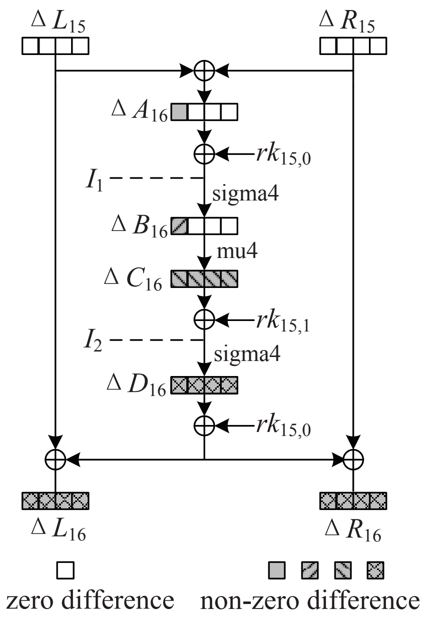

Fig. 4. Attack last round in step 2.1

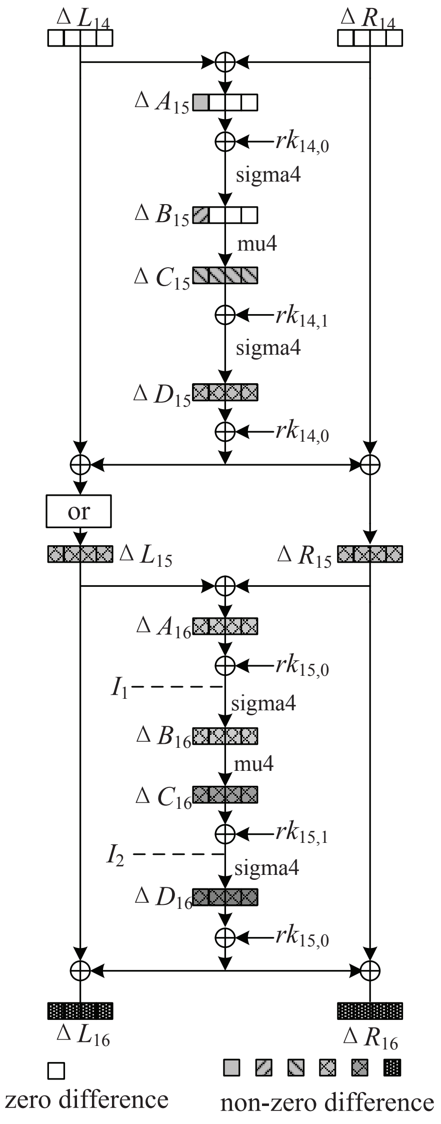

Fig. 5. Attack last round in step 2.2

Step 2.1 Recover the right input state I_2 before the second substitution layer in the last round. This is finished by considering the right ciphertext and faulty ciphertexts $c^* = c_l^* \| c_r^*$, which are obtained when faults are induced into the last round. The attack procedure is depicted in Fig.4.

On the basis of assumption of one random byte fault, ΔA_{16} has $4 \times 255 = 1020$ possibilities, so is ΔB_{16}, thus $\Delta C_{16} = \texttt{mu4}(\Delta B_{16})$ also has only 1020 possibilities.

The output difference after the second substitution layer can be calculated by $\Delta D_{16} = \Delta L_{16} \oplus \Delta L_{15} = c_l \oplus c_l^*$. For all possible values of $(\Delta C_{16}, \Delta D_{16})$, when apply differential cryptanalysis, this would lead to many possibilities of $I_2 = C_{16} \oplus rk_{15,1}$. We can further decrease the number of I_2 candidates by repeating the above method through other collected faulty ciphertexts, until the candidate set of I_2 has only one element.

Step 2.2 Recover the right input state I_1 before the first substitution layer in the last round. This is finished by considering the right input state I_2 deduced in step 2.1 and the faulty ciphertexts $c^* = c_l^* \| c_r^*$, which are obtained when faults are induced in the penultimate round. The attack procedure is depicted in Fig.5.

Since the round-key addition layer doesn't influence the difference, according to proposition 1, both ΔA_{16} and ΔD_{16} could be calculated by $\Delta L_{16} = \Delta c_l$ and $\Delta R_{16} = \Delta c_r$. Moreover, D_{16} is obtained through $D_{16} = \texttt{sigma4}(I_2)$, thus $D_{16}^* = D_{16} \oplus \Delta D_{16}$ is known, and ΔC_{16} can be obtained as

$$\begin{aligned}\Delta C_{16} = C_{16} \oplus C_{16}^* &= \texttt{sigma4}^{-1}(D_{16}) \oplus rk_{15,1} \oplus \texttt{sigma4}^{-1}(D_{16}^*) \oplus rk_{15,1} \\ &= \texttt{sigma4}^{-1}(D_{16}) \oplus \texttt{sigma4}^{-1}(D_{16}^*).\end{aligned}$$

According to $\texttt{mu4}^{-1}$, ΔB_{16} can be calculated by $\Delta B_{16} = \texttt{mu4}^{-1}(\Delta C_{16})$. After getting sufficient pairs $(\Delta A_{16}, \Delta B_{16})$ from the correct and faulty ciphertexts, we can apply the differential cryptanalysis on the first substitution layer to uniquely deduce the right input state $I_1 = A_{16} \oplus rk_{15,0}$ before this substitution layer.

Step 2.3 Recover the last round subkey $rk_{15,0}$ and $rk_{15,1}$. This is finished by directly calculate these sub-keys from the right intermediate state I_1, I_2 as described in Section 4.2.

Step 3 Recover round sub-keys from the 2-nd round to the 15-th round in reverse order. Since the last round sub-key is obtained, we can peel off the last round to obtain the output of the penultimate round. Then the same technique as described in Step 2 could be used to recover other round sub-keys. More precisely, we can do as follows:

For $i = 2, 3, \ldots, 15$, consider the right ciphertext and faulty ciphertexts when faults are induced in the $(16-i)$-th and $(17-i)$-th round, peel off the last $(i-1)$-th round(s) according to the deduced sub-key(s), and use these outputs of the $(17-i)$-th round to recover the $(17-i)$-th round subkey. This can be finished by adopting the same technique as in Step 2.

Step 4 Recover the first round sub-key. According to the recovered sub-keys from the 2-nd round to the 16-th round, obtain the outputs of the first round for the right ciphertext and faulty ciphertexts when faults are induced in the first round. Retrieve the first round subkey using the same technique as described in Section 4.2.

4.5 Complexity Analysis

To evaluate how many faults are needed to recover the whole round sub-keys, we firstly concentrate on the one round situation. It is easy to show that, to recover one round sub-key, the main complexity is dominated by step 2.1 and step 2.2.

In step 2.1, ΔC_{16} has 1020 possibilities, so to uniquely deduce all bytes of I_2 (thus D_{16}), the number of faulty ciphertexts, denoted by N, must be satisfied

$$256^4 \times \left(\frac{1020}{255^4}\right)^N \leq 1.$$

Thus in general, two faults are needed to uniquely retrieve I_2.

According to the differential property of the Sbox of FOX64, if $N_S(\alpha,\beta) \neq 0$, this difference pair (α,β) will lead to about $2.89 \approx 3$ possible inputs to the Sbox. Thus in step 2.2, when the number of faulty ciphertexts is 2, we have one passible value left for I_1 with probability

$$\mathbf{Prob} = \left(\frac{\binom{255}{2} \times \binom{255-2}{2}}{\binom{255}{2}^2}\right)^4 \approx 93.88\%.$$

The above analysis indicates that, in most cases, about 4 faults are enough to uniquely deduce the round sub-key (excluding the first round). Note that, to recover the first round subkey, we have to use the same technique as in [2], i.e. we must induce about $8 \sim 31$ faults. Thus, to retrieve the whole round sub-keys, we need $2 \times 16 + 6 = 38$ faults in the best case.

5 Experimental Results

Our proposed improved fault attack against the last round of FOX64 has been successfully implemented through computer simulation, and the fault injection is simulated by computer software. We implement the attack procedure in C++ code and execute it on a PC with Intel Pentium 1.80 GHz processor. We repeat the attack 10000 times and the results are shown in Fig.6.

From Fig.6, it is observed that step 2.1 requires from 2 to 5 faults (2.11 on average), while step 2.2 requires from 2 to 7 faults (2.14 on average). The complete attack requires from 4 to 9 faults and the average value is 4.25. A comparison between our proposed and the previous fault attack [2] against the last round of FOX64 is shown in Table 2.

Table 2 shows that to derive the last round subkey, our improved fault attack requires less faulty ciphertexts, an average of about 4.25 faults compared to 11.45 faults required in the previous fault attack [2]. Thus to recover the whole round sub-keys, our fault attack need about $4.25 \times 8 + 11.45 = 45.45$ (38 in the best case) faults on average, while about $11.45 \times 16 = 183.20$ (128 in the best case) faults on average are needed using the method in [2]. This is shown in Table 3.

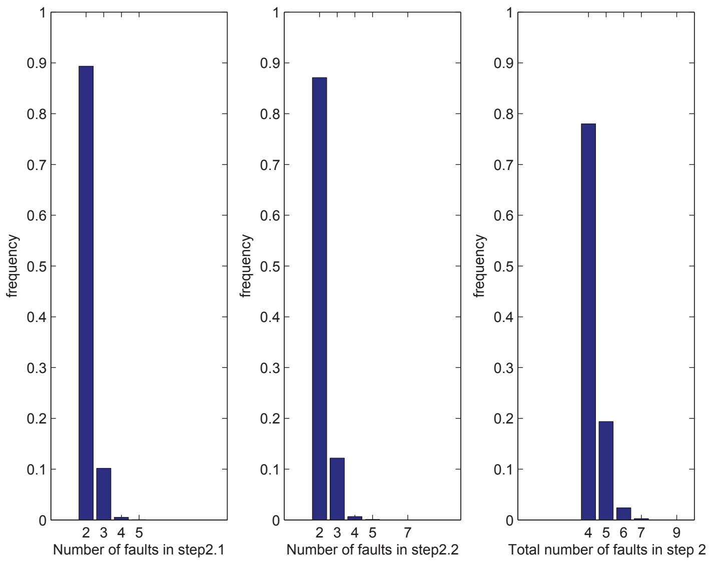

Fig. 6. Simulation results of the improved fault attack against the last round of FOX64

Table 2. Comparison with fault attacks against the last round of FOX64

Fault Location	Min	Avg	Max	Attack
Last round	8	11.45	31	[2]
Last and penultimate round	4	4.25	9	Section 4

Table 3. Comparison with fault attacks against the whole rounds of FOX64

Fault Location	Best Case	Average	Attack
Each round	128	183.20	[2]
Each round	38	45.45	Section 4

6 Conclusion

We propose an improved fault attack on FOX64. In the improved attack, one round subkey can be deduced with 4.25 faults on average, and the whole cipher can be broken with 45.45 faults on average. The technique of the proposed attack in this paper can also be easily extended to other series of FOX.

Acknowledgments

The work in this paper is supported by the Natural Science Foundation of China (No:60803156) and the open research fund of State Key Laboratory of Information Security(No: 01-07).

References

1. Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. CRYPTO 97, LNCS 1294, pp. 513-525, Springer-Verlag, 1997.
2. L. Breveglieri, I. Koren, and P. Maistri. A Fault Attack Against the FOX Cipher Family. FDTC 2006, LNCS 4236, pp. 98-105, Springer-Verlag 2006.
3. Johannes Blömer and Jean-Pierre Seifert. Fault Based Cryptanalysis of the Advanced Encryption Standard (AES). FC 2003, LNCS 2742, pp. 162-181, Springer-Verlag, 2003.
4. Chien-Ning Chen and Sung-Ming Yen. Differential Fault Analysis on AES key Schedule and Some Countermeasures. ACISP 2003, LNCS 2727, pp. 118-129, Springer-Verlag, 2003.
5. Pierre Dusart, Gilles Letourneux and Olivier Vivolo. Differential Fault Analysis on A.E.S. ACNS 2003, LNCS 2846, pp. 293-306, Springer-Verlag, 2003.
6. Christophe Giraud. DFA on AES. AES 2004, LNCS 3373, pp. 27-41, Springer-Verlag, 2005.
7. P Junod, S Vaudenay. FOX: A New Family of Block Ciphers . SAC 2004, LNCS 3357, pp. 114-129, 2005.
8. Chong Hee Kim and Jean-Jacques Quisquater. New Differential Fault Analysis on AES Key Schedule: Two Faults Are Enough. CARDIS 2008, LNCS 5189, pp. 48-60, Springer-Verlag, 2008.
9. Debdeep Mukhopadhyay. An Improved Fault Based Attack of the Advanced Encryption Standard. Africacrypt 2009, LNCS 5580, pp. 421-434, 2009.
10. Amir Moradi, Mohammad T. Manzuri Shalmani, and Mahmoud Salmasizadeh. A Generalized Method of Differential Fault Attack Against AES Cryptosystem. CHES 2006, LNCS 4249, pp. 91-100, Springer-Verlag, 2006.
11. Gilles Piret and Jean-Jacques Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. CHES 2003, LNCS 2779, pp. 77-88, Springer-Verlag, 2003.
12. Junko Takahashi, Toshinori Fukunaga, Kimihiro Yamakoshi. DFA Mechanism on the AES Key Schedule. FDTC 2007, pp. 62-74, IEEE Computer Society, 2007.