Computers, Materials & Continua CMC, vol.65, no.2, pp.1425-1436, 2020

An Improved Differential Fault Analysis on Block Cipher KLEIN-64

Min Long[1, *], Man Kong[1], Sai Long[1] and Xiang Zhang[2]

Abstract: KLEIN-64 is a lightweight block cipher designed for resource-constrained environment, and it has advantages in software performance and hardware implementation. Recent investigation shows that KLEIN-64 is vulnerable to differential fault attack (DFA). In this paper, an improved DFA is performed to KLEIN-64. It is found that the differential propagation path and the distribution of the *S*-box can be fully utilized to distinguish the correct and wrong keys when a half-byte fault is injected in the 10^{th} round. By analyzing the difference matrix before the last round of *S*-box, the location of fault injection can be limited to a small range. Thus, this improved analysis can greatly improve the attack efficiency. For the best case, the scale of brute-force attack is only 256. While for the worst case, the scale of brute-force attack is far less than 2^{32} with another half byte fault injection, and the probability for this case is 1/64. Furthermore, the measures for KLEIN-64 in resisting the improved DFA are proposed.

Keywords: Block cipher, KLEIN-64, differential fault analysis, half-byte fault injection.

1 Introduction

Information security is an important research topic in information field. As an important security technology, encryption has been constantly improved and matured for protecting data. Currently, encryption is closely related to our daily life, such as embedded systems, wireless sensors, RFID and other devices, which are widely used in bank cards, bus cards, access control cards, and etc. These situations require fast information processing. Classical encryption algorithms can be processed in parallel to improve the real-time performance of encryption [Min, Yang and Wang (2019)]. Due to the strict requirements on circuits size, computing power and storage space of cryptographic devices, lightweight block cipher has emerged and developed. Common lightweight cryptographic algorithms include LBlock [Wu and Zhang (2011)], LED [Guo, Peyrin, Poschmann et al. (2011)], PRINCE [Borghoff, Canteaut, Güneysu et al. (2012)]. And there are some recent proposed methods, such as CSL [Lamkuche and Pramod (2020)], SFN [Li, Liu, Zhou et al. (2018)]. A typical lightweight block cipher must address three challenges: minimal overhead, low-power consumption and adequate security capability [Liu, Wang, Chaudhry et al. (2018)]. As a typical lightweight block cipher, KLEIN-64 was proposed

[1] Changsha University of Science and Technology, Changsha, 410014, China.

[2] School of Computing, National University of Singapore, Singapore.

[*] Corresponding Author: Min Long. Email: caslongm@aliyun.com.

Received: 20 April 2020; Accepted: 03 June 2020.

by Gong et al. [Gong, Nikova and Law (2011)] in RFIDsec. It is not only suitable for hardware resource-constrained environment, but also has good performance in software implementation.

At present, many attacks are proposed to compromise cryptographic algorithms. Among them, DFA is a bypass attack that uses an operation error in the case of interference to restore the secret key. Thereafter, DFA has been widely used in public key cryptosystems and block ciphers, which imposes a serious threat to the security of many ciphers.

According to the structure of lightweight cryptographic algorithms, there are various models of DFA. Jeong et al. [Jeong, Lee and Lim (2013)] have found that injecting random nibble faults to the input register of round 29 can recover the secret key of LBlock by using an exhaustive search of 2^{30} and seven random nibble fault injections on average. Vafaei et al. [Vafaei, Bagheri, Saha et al. (2018)] have revealed that DFA for random half-byte fault injection of SKINNY, and the extraction of the master key of SKINNY requires about 10 nibble fault injections in an average for 64-bit version of the cipher. Wang et al. [Wang, Zhang, Wang et al. (2018)] proposed that a DFA of MIBS cipher by injecting two faults in the last round, and the data complexity is only 2^{17}. Later, Gao et al. [Gao, Wang, Yuan et al. (2019)] have found that probabilistic analysis of differential fault attack on MIBS was proposed by a new method of differential fault attack, which based on the nibble-group differential diffusion property. Zhang et al. [Zhang, Wu, Li et al. (2019)] proposed that 64 bits of AES-128 key can be recovered by inducing a random two-byte fault in the first column of the 9^{th} round key with discontinuous rows. Wei et al. [Wei, Rong and Fan (2018)] have found that a DFA of LBlock based on a nibble oriented random fault model was presented by injecting faults in the 27^{th} round to the 29^{th} round, an average of 13.3 faults are needed for revealing the whole key information. Gruber et al. [Gruber and Selmke (2019)] proposed that a DFA of KLEIN-64 is introduced. It determines the last round of keys with one byte random fault and at least five fault ciphertext, and the key space is reduced from 64 bit to 32 bit. Liao et al. [Liao, Cui, Liao et al. (2017)] have found that an efficient differential analysis algorithm for random faults of AES. The actual results show that the algorithm can recover the key by analyzing only 2.17 fault ciphertexts on average. Fu et al. [Fu, Xu and Pan (2016)] discussed the possible injection positions with different number of faults of ITUbee. The most efficient attack takes 2^{25} round function operations with 4 faults, which is achieved in a few seconds on a PC.

The main attack idea of this paper is derived from [Gruber and Selmke (2019)]. In the proposed attack, if a random nibble fault is injected into the register in the 10^{th} round, it is necessary to exhaust at least 2^8 to recover the subkeys of the last round, and the initial key can be obtained by the inverse key schedule algorithm with this analysis.

The rest of this paper is organized as follows. In Section 2, the basic principle of KLEIN-64 is briefly introduced, the details of the attack are described in Section 3. Some measures of improving the security are provided in Section 4. Finally, some conclusions are drawn in Section 5.

2 Preliminary knowledge on KLEIN-64

The general structure of KLEIN-64 is shown in Algorithm 1, and it mainly consists of

four parts: AddRoundKey, SubNibble, RotateNibble and MixNibble.

Algorithm 1. The structure of the KLEIN cipher

$Sk^1 \leftarrow KEY$
STATE ← PLAINTEXT
for i = 1 to R do
AddRoundKey(STATE,sk^i)
SubNibbles(STATE)
RotateNibbles(STATE)
MixNibbles(STATE)
$Sk^{i+1} \leftarrow$ KeySchedule(sk^i,i)
end for
CIPHERTEXT ← AddRoundKey(STATE,sk^{R+1})

2.1 AddRoundKey

KLEIN-64 starts from AddRoundKey and ends with AddRound-Key. Therefore, it is not easy to carry out inverse operation when the key is unknown. The plaintext is denoted by a 4×4 matrix *P*, and it is represented as

$$P = \begin{pmatrix} p[0] & p[1] & p[8] & p[9] \\ p[2] & p[3] & p[10] & p[11] \\ p[4] & p[5] & p[12] & p[13] \\ p[6] & p[7] & p[14] & p[15] \end{pmatrix} \tag{1}$$

The key is denoted by a 4×4 matrix *K* as

$$K = \begin{pmatrix} k[0] & k[1] & k[8] & k[9] \\ k[2] & k[3] & k[10] & k[11] \\ k[4] & k[5] & k[12] & k[13] \\ k[6] & k[7] & k[14] & k[15] \end{pmatrix} \tag{2}$$

2.2 SubNibbles

Table 1: The 4-bit S-box used in KLEIN

Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	7	4	A	9	1	F	B	0	C	3	2	6	8	E	D	5

In SubNibbles, the results after XOR operation will be divided into 16 4-bit nibbles, and they are used as the inputs of 16 S-boxes. The S-box *S* of KLEIN-64 is a 4×4 involutive permutation. The non-linear permutation executed by *S* is described in Tab. 1.

2.3 RotateNibbles

As shown in Fig. 1, 16 nibbles s_0^i, s_1^i, ..., s_{15}^i are rotated to the left by two bytes per round, and 16 4-bit state data r_0^i, r_1^i, ..., r_{15}^i can be obtained. The inverse operation is simply rotated to the right by two bytes per round.

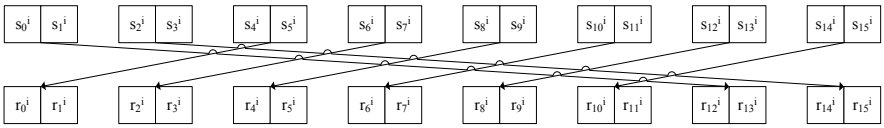

Figure 1: The rotate nibbles step

2.4 MixNibbles

The input states are divided into left and right parts. Every 8-bit forms an element of the matrix, so the left and the right can be viewed as two column vectors. The intermediate state C is obtained by multiplying an MDS matrix M with two column vectors, respectively. The input nibbles of the i-th round are divided in two tuples. They are proceeded by an MDS matrix M, which are represented as

$$C^{i+1} = \begin{pmatrix} c_0^{i+1} \| c_1^{i+1} & c_8^{i+1} \| c_{91}^{i+1} \\ c_2^{i+1} \| c_3^{i+1} & c_{10}^{i+1} \| c_{11}^{i+1} \\ c_4^{i+1} \| c_5^{i+1} & c_{12}^{i+1} \| c_{13}^{i+1} \\ c_6^{i+1} \| c_7^{i+1} & c_{14}^{i+1} \| c_{15}^{i+1} \end{pmatrix} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} r_0^{i+1} \| r_1^{i+1} & r_8^{i+1} \| r_9^{i+1} \\ r_2^{i+1} \| r_3^{i+1} & r_{10}^{i+1} \| r_{11}^{i+1} \\ r_4^{i+1} \| r_5^{i+1} & r_{12}^{i+1} \| r_{13}^{i+1} \\ r_6^{i+1} \| r_7^{i+1} & r_{14}^{i+1} \| r_{15}^{i+1} \end{pmatrix} \tag{3}$$

2.5 Key schedule

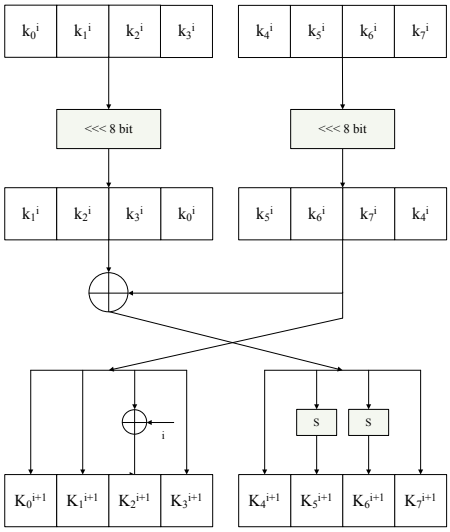

Figure 2: The key schedule algorithm of 64-bit key length

The key schedule algorithm is illustrated in Fig. 2. A 64-bit master key is denoted by k_0^i, ..., k_7^i (i is the round counter), and it is divided in two parts.

3 The proposed DFA on KLEIN-64

Here are 3 assumptions for the proposed DFA:
(1) The attacker has the capability of specifying a plaintext for encryption and obtaining the corresponding ciphertext;
(2) The attacker can induce a random nibble fault in the 10^{th} round;
(3) The fault location and value are both unknown.

3.1 Design inspiration and related modified work

KLEIN-64 is designed by half byte except nonlinear transformation. Other steps are converted in bytes. Gruber et al. [Gruber and Selmke (2019)] have found that the attack on KLEIN-64 is based on one byte fault injection, which will be more convenient in the analysis. However, when a fault is introduced in the intermediate state, four-byte subkeys including 32 bits in the last round need to be exhausted. This paper is to inject a random half byte fault into the 10^{th} round and exhaust the remaining 8 half bytes in the last round. In the worst case, the scale of brute-force attack is 2^{32}, which is the same as that [Gruber and Selmke (2019)], and the probability of this situation is 1/64. In the best case, the scale of brute-force attack is only 2^8.

Algorithm 2. The structure of the KLEIN-64 cipher

```
Sk^1 ← KEY
STATE ← PLAINTEXT
for i = 1 to R-1 do
   AddRoundKey(STATE,sk^i)
   SubNibbles(STATE)
   RotateNibbles(STATE)
   MixNibbles(STATE)
   Sk^(i+1) ← KeySchedule(sk^i,i)
end for
   AddRoundKey(STATE,sk^R)
   SubNibbles(STATE)
   RotateNibbles(STATE)
   Sk^(R+1) ← KeySchedule(sk^R, R)
   AddRoundKey(STATE, invMixNibble(sk^(R+1)))
   CIPHERTEXT ←MixNibble(STATE)
```

In encryption, the column mixing transformation is a linear matrix transformation, and the MDS matrix is invertible. It means that $A \oplus M(B)=M(M^{-1}(A) \oplus B)$ always holds. Therefore, the encryption process can be changed without changing the encryption results.

As shown in Algorithm 2, the inverse column transformation is first performed on the key of the last round, and then XOR the results of row shift in the last round. Thereafter, column mixing transformation is performed on it. The final encryption results are consistent with the encryption results before modification.

3.2 Difference propagation path

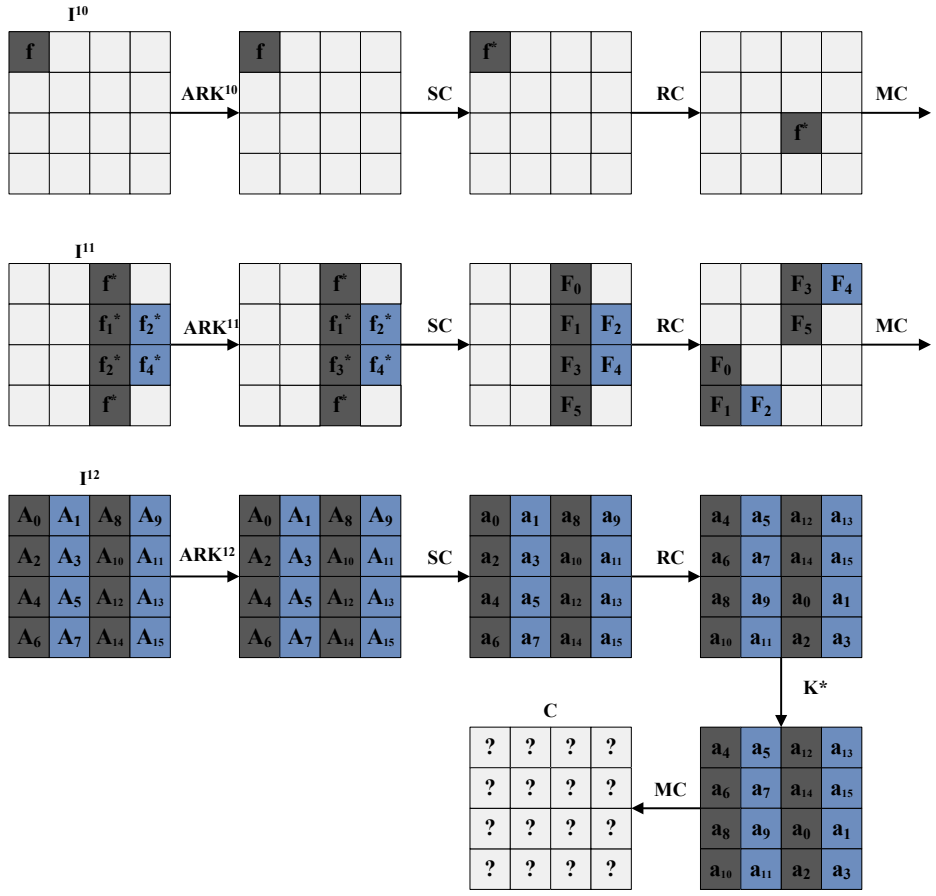

Figure 3: The diffusion process

In this attack, it is supposed that a half-byte fault is injected at the register I10 in the 10th round, and the input difference is f, and the output difference is f^*. Pairs (C, C^*) is the correct ciphertext and the fault ciphertext with a same length of 64 bits, respectively.

The diffusion process is shown in Fig. 3. The gray part denotes that the value of difference is stable, and the blue part indicates the uncertainty of the difference. If the XOR results of the first bit of the status byte and the first bit of the fault is 0, it means that the difference is 0000; otherwise, it means that the difference is 1011. This difference comes from the column mixing transformation. That is, when a value is multiplied by 2, the result shifts one bit to the left. If the leftmost bit of the value is 1, it will XOR 0X1B after the shift. According to the column mixing transformation of KLEIN-64, the following conclusions can be drawn.

(1) When the fault only occurs in the four higher bits, the whole fault byte is $((3f^*+1)\|b)$ by multiplying 3 or $((2f^*+1)\|b)$ by multiplying 2, where b denotes the difference 0000 or 1011, and || denotes concatenation.
(2) When the fault only occurs in the four lower bits, the whole fault byte is $(1\|(3f^*))$ multiplying 3 or $(1\|2f^*)$ multiplying 2, where the four higher bits cannot be determined.
(3) When the fault occurs both in four higher bits and four lower bits, the whole fault byte is $((3f_1^*+1)\|(3f_1^*+b))$ multiplying 3 or $((2f_1^*+1)\|(2f_1^*+b))$ multiplying 2.
The K^* in Figure 3 denotes the inverse column mixing transformation on the secret key in the last round, and $K^*=Invmixnibble$ (RK13).

3.3. Description of attack

According to Fig. 3, since RK13 can be calculated by K^*, the attack is to recover K^*. The attack is described in the following.

(1) Collect correct ciphertext. Select plaintext P for encryption and obtain the correct ciphertext C.

(2) Collect fault ciphertext. Encrypt the same plaintext P, and obtain the fault ciphertext C^* by injecting a random nibble fault in the 10th round.

(3) Get the results of inverse column transformation. For each pair (C, C^*), compute $X=MC^{-1}(C)$ and $X^*=MC^{-1}(C^*)$.

(4) Analyze column mixing transformation in the 10th round. After column mixing transformation in the 10th round, the difference f_2^* and f_4^* is 0 or B. From 3.2, it can be determined whether f_1^* and f_3^* needs to be XORed 1 or not.

(5) Analyze column mixing transformation in the 11th round. After column mixing transformation in the 11th round, the difference $A_1, A_3, A_5, A_7, A_9, A_{11}, A_{13}, A_{15}$ is 0 or B. From 3.2, it can be determined whether $A_0, A_2, A_4, A_6, A_8, A_{10}, A_{12}, A_{14}$ need to be XORed 1 or not.

(6) Calculate the candidate value of K^*. Since S-box is an involutive permutation, $S[X]=S^{-1}[X]$ can be obtained.

(a) Guess $K^*[12,14,0,2]$. Because $X[12]$, $X[14]$, $X[0]$ and $X[2]$ are all known, their differences can be obtained. (The bold indicates that it may or may not exist. If b is 1, it means it exists, otherwise, it does not exist.)

$$A_0=F_0\oplus F_1=S(X[12]\oplus K^*[12])\oplus S(X[12]\oplus a_{12}\oplus K^*[12]) \quad (4)$$

$$A_2=3F_0\oplus F_1\oplus \mathbf{1}=S(X[14]\oplus K^*[14])\oplus S(X[14]\oplus a_{14}\oplus K^*[14]) \quad (5)$$

$$A_4=2F_0\oplus 3F_1\oplus \mathbf{1}\oplus \mathbf{1}=S(X[0]\oplus K^*[0])\oplus S(X[0]\oplus a_0\oplus K^*[0]) \quad (6)$$

$$A_6=F_0\oplus 2F_1\oplus \mathbf{1}=S(X[2]\oplus K^*[2])\oplus S(X[2]\oplus a_2\oplus K^*[2]) \quad (7)$$

(b) Guess $K^*[13,15,1,3]$. Because $X[13]$, $X[15]$, $X[1]$ and $X[3]$ are known, their differences can be obtained. The bold indicates that it may or may not exist. When the difference value equals the output difference with the input difference b of S-box, b exists; otherwise, it does not exist.

$$A_1=F_2=\boldsymbol{b}=S(X[13]\oplus K^*[13])\oplus S(X[13]\oplus a_{13}\oplus K^*[13]) \quad (8)$$

$$A_3=\boldsymbol{b}\oplus F_2=\boldsymbol{b}\oplus \mathbf{b}=S(X[15]\oplus K^*[15])\oplus S(X[15]\oplus a_{15}\oplus K^*[15]) \quad (9)$$

$$A_5=\boldsymbol{b}\oplus 3F_2\oplus\boldsymbol{b}=\boldsymbol{b}\oplus\boldsymbol{3b}\oplus\boldsymbol{b}=S(X[1]\oplus K^*[1])\oplus S(X[1]\oplus a_1\oplus K^*[1]) \quad (10)$$

$$A_7=2F_2\oplus\boldsymbol{b}=\boldsymbol{2b}\oplus\boldsymbol{b}=S(X[3]\oplus K^*[3])\oplus S(X[3]\oplus a_3\oplus K^*[3]) \quad (11)$$

(c) Guess $K^*[4,6,8,10]$. Because $X[4]$, $X[6]$, $X[8]$ and $X[10]$ are known, their differences can be obtained. (The bold indicates that it may or may not exist. If b is 1, it means it exists; otherwise, it does not exist.)

$$A_8=2F_3\oplus\boldsymbol{1}\oplus 3F_5\oplus\boldsymbol{1}=S(X[4]\oplus K^*[4])\oplus S(X[4]\oplus a_4\oplus K^*[4]) \quad (12)$$

$$A_{10}=F_3\oplus 2F_5\oplus\boldsymbol{1}=S(X[6]\oplus K^*[6])\oplus S(X[6]\oplus a_6\oplus K^*[6]) \quad (13)$$

$$A_{12}=F_3\oplus F_5=S(X[8]\oplus K^*[8])\oplus S(X[8]\oplus a_8\oplus K^*[8]) \quad (14)$$

$$A_{14}=3F_3\oplus\boldsymbol{1}\oplus F_5=S(X[10]\oplus K^*[10])\oplus S(X[10]\oplus a_{10}\oplus K^*[10]) \quad (15)$$

(d) Guess $K^*[5,7,9,11]$. Because $X[5]$, $X[7]$, $X[9]$ and $X[11]$ are known, their differences can be obtained. (The boldface indicates that it may or may not exist. When the difference value equals the output difference with the input difference b of S-box, b exists; otherwise, it does not exist.

$$A_9=2F_4\oplus\boldsymbol{b}\oplus\boldsymbol{b}=\boldsymbol{2b}\oplus\boldsymbol{b}\oplus\boldsymbol{b}=S(X[5]\oplus K^*[5])\oplus S(X[5]\oplus a_5\oplus K^*[5]) \quad (16)$$

$$A_{11}=F_4\oplus\boldsymbol{b}=\boldsymbol{b}\oplus\boldsymbol{b}=S(X[7]\oplus K^*[7])\oplus S(X[7]\oplus a_7\oplus K^*[7]) \quad (17)$$

$$A_{13}=F_4=\boldsymbol{b}=S(X[9]\oplus K^*[9])\oplus S(X[9]\oplus a_9\oplus K^*[9]) \quad (18)$$

$$A_{15}=3F_4\oplus\boldsymbol{b}=\boldsymbol{3b}\oplus\boldsymbol{b}=S(X[11]\oplus K^*[11])\oplus S(X[11]\oplus a_{11}\oplus K^*[11]) \quad (19)$$

(e) Calculate the candidates of nibble K^* in (a) and (c). Half-byte keys that do not satisfy the equation can be discarded in (a) and (c), and the remaining keys can be exhausted. When another half byte fault is injected, the half byte secret keys that does not satisfy the equation are discarded. In this way, the scale of brute-force attack is far less than 2^{32}.

(f) Calculate the candidates of nibble K^* in (b) and (d). If the difference in (b) and (d) is 0, the exhaustion of guessing the half-byte keys is 2^{32}. However, it can be reduced by combining (16) (here, another nibble fault injection is needed), and this happens with a probability of 1/ 64. If the difference value in (b) and (d) is not zero, the input of the S box can be derived from the known difference of the input and the output of S box. The number of elements in the input set is 2 or 4. MC^{-1} (C) XOR the results of S box and the row shifting, and the candidate values of half-byte keys in (b) and (d) can also be obtained. The exhaustion of it is at least 2^8.

In the difference matrix before the last round of the S-box, it can be found:

(i) If it is 0 or b in the 2nd and 4th column of the matrix, the fault is possibly injected at I_0^{10}, I_2^{10}, I_4^{10}, I_6^{10}, I_8^{10}, I_{10}^{10}, I_{12}^{10}, or I_{14}^{10}.

(ii) If it is 0 or b in the 1th and 3rd column of the matrix, the fault is possibly injected at I_1^{10}, I_3^{10}, I_5^{10}, I_7^{10}, I_9^{10}, I_{11}^{10}, I_{13}^{10}, or I_{15}^{10}.

(iii) In the case of (1), according to the coefficient relationship between $F0$ and $F1$ in (a), $F3$ and $F5$ in (c), the location of fault injection is limited in the part of I_0^{10}, I_2^{10}, I_{12}^{10}, I_{14}^{10} or the part of I_4^{10}, I_6^{10}, I_8^{10}, I_{10}^{10}.

Finally, the location can be exhausted only for 4 times.

4 The improvement of KLEIN-64

In the proposed DFA, it takes the column mixing step in KLEIN-64 as a breakthrough to reduce the search space of secret keys, and the non-uniform difference distribution of the *S*-box also help finding the candidate secret keys. In this section, some improvements are made to resist the proposed DFA from two aspects.

4.1 Improved MixNibbles

In the original MixNibbles, the intermediate state C is obtained by multiplying an MDS matrix M with two column vectors, respectively. As the MDS matrix is invertible, $A \oplus M(B)=M(M^{-1}(A) \oplus B)$ always holds. Therefore, the change of the encryption process will not change the encryption results. Here, let the intermediate state of the *S*-box multiply the matrix M, the results have a modular arithmetic X^4+1. Then, the cipher state can be conceptually arranged in a 4×4 grid, where each nibble represents an element from $GF(2^4)$. The matrix M is

$$M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \tag{20}$$

With the above operations, it cannot find the rule of reducing the complexity of secret key through the second half byte of each byte state in the improved algorithm.

4.2 Improved S-box

KLEIN-64 adopts the *S*-box with 4 in and 4 out, which is essentially a multi-output function from $GF(2)^4$ to $GF(2)^4$. The difference distribution of the S-box is listed in Tab. 2. From Tab. 2, it can be seen that the number of non-zero in each row is not the same. The non-uniformity of the distribution of the input difference and the output difference leads to the possibility of differential analysis. Here, a new *S*-box is constructed to improve the capability of resisting DFA.

(i) x^4+x+1, x^4+x^3+1, and $x^4+x^3+x^2+1$ are irreducible polynomials in $GF(2^4)$. Take one of them, such as x^4+x+1, for an example, find the inverse of the corresponding state half-byte [0, 1, 2, 3, 4, 5, 6, 7, 8, 9 *A*, *B*, *C*, *D*, *E*, *F*] : [0, 9, *E*, *D*, *B*, *7*, 6, *F*, 2, *C*, 5, *A*, 4, 3, 8].

(ii) Affine transformation is performed on the result of (i) as follows

$$\begin{bmatrix} Y_3 \\ Y_2 \\ Y_1 \\ Y_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_3 \\ X_2 \\ X_1 \\ X_0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}. \tag{21}$$

Table 2: The difference distribution of s-box of KLEIN algorithm

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	4	2	0	2	0	2	0	0	2	0	0	2	2
2	0	0	0	0	0	4	0	0	0	0	2	2	0	4	2	2
3	0	4	0	2	2	0	0	0	0	0	2	0	0	2	4	0
4	0	2	0	2	2	0	2	0	0	2	0	2	0	2	0	2
5	0	0	4	0	0	2	0	2	2	0	2	4	0	0	0	0
6	0	2	0	0	2	0	4	0	2	0	2	2	0	2	0	0
7	0	0	0	0	0	2	0	2	2	2	0	0	2	0	4	2
8	0	2	0	0	0	2	2	2	2	2	0	2	0	0	0	2
9	0	0	0	0	2	0	0	2	2	0	0	2	2	2	2	2
A	0	0	2	2	0	2	2	0	0	0	4	0	2	0	2	0
B	0	2	2	0	2	4	2	0	2	2	0	0	0	0	0	0
C	0	0	0	0	0	0	2	2	0	2	2	0	4	2	0	2
D	0	0	4	2	2	0	0	0	0	2	0	0	2	2	0	2
E	0	2	2	4	0	0	0	4	0	2	2	0	0	0	0	0
F	0	2	2	2	0	0	0	2	2	2	0	0	2	2	0	0

Table 3: The improved S-box

Input	0	1	2	3	4	5	6	7	8	9	*A*	*B*	*C*	*D*	*E*	*F*
Output	2	*F*	4	0	3	*A*	6	*B*	*D*	*C*	*E*	8	7	5	1	9

Table 4: The improved s-box difference distribution

	0	1	2	3	4	5	6	7	8	9	*A*	*B*	*C*	*D*	*E*	*F*
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	2	2	0	2	0	2	0	2	2	0	0	0	4	0	0
2	0	2	0	2	2	2	4	0	0	0	0	0	2	0	0	2
3	0	0	4	0	2	2	0	0	2	0	0	2	2	0	2	0
4	0	4	2	0	0	2	0	0	0	2	2	2	0	0	0	2
5	0	0	0	0	0	0	2	2	4	2	0	2	0	2	0	2
6	0	0	0	0	4	2	0	2	0	2	2	0	2	2	0	0
7	0	0	0	2	2	0	0	0	0	4	0	2	0	2	2	2
8	0	0	2	2	2	0	0	2	2	0	2	0	0	0	0	4
9	0	0	2	0	0	0	2	0	0	0	2	0	2	2	4	2
A	0	2	2	2	0	0	0	2	0	2	0	0	4	0	2	0
B	0	2	0	2	0	0	0	0	2	0	4	2	2	2	0	0
C	0	0	0	2	0	4	2	0	2	2	2	0	0	0	2	0
D	0	2	0	0	0	2	0	4	2	0	0	0	0	2	2	2
E	0	0	2	4	0	2	2	2	0	0	0	2	0	2	0	0
F	0	2	0	0	2	0	2	2	0	0	2	4	0	0	2	0

The affine transformation is expressed as

$$Y(x)=\upsilon(x)+\left(X^{-1}\right)\cdot\mu(x)\bmod m(x). \tag{22}$$

Here, it needs to discuss how to select polynomials $v(x)$, $u(x)$, and $m(x)$ in $GF(2^4)$, where $m(x)$ should be a polynomial with the highest degree of 4. $m(x)=x^4+1$ is a reducible polynomial with simple form in $GF(2^4)$, and $u(x)$ is arbitrarily chosen and it is prime with $m(x)$. Here, $u(x)=x^3+x+1$. $V(x)$ is an affine constant to ensure that there is no fixed point or inverse fixed point in the *S*-box and $v(x)=x$.

Through the above steps, the *S*-box is calculated by Tab. 3 and its difference distribution is listed in Tab. 4. For this *S*-box, the capability of resisting DFA essentially depends on the differential distribution and differential uniformity. Since all the number of non-zero in each row are 7, and it is evenly distributed, which can improve the capability of resisting DFA.

5 Conclusion

In this paper, an improved DFA is successfully launched on KLEIN-64 by half-byte fault injection. In the proposed attack, the fault location can be limited to 4 positions, and half of the whole key can be exhausted by another random half-byte fault, which significantly reduces the exhaustion scale. Another part of half-byte of the key can be guessed by 2^8 at the best case. Even at the worst case, the scale of brute-force attack is far less than 2^{32} with another half byte fault injection, and this case happens with a probability of 1/64. In order to enhance the capability of resisting the differential attack, some measures are also proposed in the design of mix nibbles part and S-box in the original algorithm.

Funding Statement: This work was supported in part by project supported by National Natural Science Foundation of China (Grant Nos. U1936115, 61572182).

Conflicts of Interest: Authors declare that they have no conflicts of interest to report regarding the present study.

References

Borghoff, J.; Canteaut, A.; Güneysu, T.; Kavun, B.; Knezevic, M. (2012): PRINCE-a low-latency block cipher for pervasive computing applications. *ASIACRYPT 2012, LNCS, International Association for Cryptologic Research*, vol. 7658, pp. 208-225.

Fu, S.; Xu, G. A.; Pan, J.; Wang, Z. Y.; Wang, A. (2016): Differential fault attack on ITUbee block cipher. *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 2, pp. 1-10.

Gao, Y.; Wang, Y. J.; Yuan, Q. J.; Wang, T.; Wang, X. B. (2019): Probabilistic analysis of differential fault attack on MIBS. *The Institute of Electronics, Information and Communication Engineers*, vol. E102-D, no. 2, pp. 299-306.

Gong, Z.; Nikova, S. L.; Law, Y. W. (2011): KLEIN: a new family of lightweight block ciphers. RFIDSec, Amherst, USA, pp. 1-18.

Gruber, M.; Selmke, B. (2019): Differential fault attack on KLEIN. *Constructive Side-Channel Analysis and Secure Design*, pp. 80-95, Darmstadt, Germany.

Guo, J.; Peyrin, T; Poschmann, A.; Tobshaw, M. (2011): The LED block cipher.

International Workshop on Cryptographic Hardware and Embedded Systems, pp. 326-341, Nara, Japan.

Jeong, K.; Lee, C. (2012): Differential fault analysis on block cipher LED-64, *Lecture Notes in Electrical Engineering*, *Future Information Technology, Application, and Service*, pp. 747-755, Seoul, Korea.

Jeong, K.; Lee, C.; Lim, J. I. (2013): Improved differential fault analysis on lightweight block cipher LBlock for wireless sensor networks. *EURASIP Journal on Wireless Communications and Networking*, vol. 151, no. 1, pp. 1-9.

Lamkuche, H. S.; Pramod, D. (2020): CSL: FPGA implementation of lightweight block cipher for power-constrained devices. *International Journal of Information and Computer Security*, vol. 12, no. 2/3, pp. 349-377.

Li, L.; Liu, B. T.; Zhou, Y. M.; Zou, Y. (2018): SFN: a new lightweight block cipher. *Microprocessors and Microsystems*, vol. 60, no. 9, pp. 138-150.

Liao, N.; Cui, X. X.; Liao, K.; Wang, T.; Yu, D. S. et al. (2017): Improving DFA attacks on AES with unknown and random faults. *Science China Information Sciences*, vol. 60, no. 4, pp. 1-14.

Liu, P. C.; Wang X. J.; Chaudhry, S. R.; Javeed, K.; Ma, Y. et al. (2018): Secure video streaming with lightweight cipher PRESENT in an SDN testbed. *Computers, Materials & Continua*, vol. 57, no. 3, pp. 353-363.

Min, Z.; Yang, G.; Wang, J.; Kim, G. J. (2019): A privacy-preserving BGN-type parallel homomorphic encryption algorithm based on LWE, *Journal of Internet Technology*, vol. 20, no. 7, pp. 2189-2200.

Vafaei, N.; Bagheri, N.; Saha, S.; Mukhopadhyay, D. (2018): Differential fault attacks on SKINNY block cipher. *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp. 177-197, Kanpur, India.

Wang, Y.; Zhang, S.; Wang, T.; Cao, Y. (2018): Differential fault attack on block cipher MIBS, J. *University of Electronic Science and Technology of China*, vol. 47, no. 4, pp. 601-605.

Wei, Y.; Rong, Y.; Fan, C. (2018): Differential fault attacks on lightweight cipher lblock. *Fundamenta Informatica*e, vol. 157, no. 1-2, pp. 125-139.

Wu, W.; Zhang, L. (2011): LBlock: a lightweight block cipher. *9th International Conference on Applied Cryptography and Network Security*, Nerja, Spain, pp. 327-344.

Zhang, J.; Wu, N.; Li, J; Zhou, F. (2019): A novel differential fault analysis using two byte fault model on AES Key schedule". *IET Circuits Devices & Systems*, vol. 13, no. 5, pp. 661-666.