Cybersecurity

RESEARCH

Open Access

The differential fault analysis on block cipher FeW

Haiyan Xiao[1,2], Lifang Wang[1] and Jinyong Chang[3,4*]

Abstract

Feather weight (FeW) cipher is a lightweight block cipher proposed by Kumar et al. in 2019, which takes 64 bits plaintext as input and produces 64 bits ciphertext. As Kumar et al. said, FeW is a software oriented design with the aim of achieving high efficiency in software based environments. It seems that FeW is immune to many cryptographic attacks, like linear, impossible differential, differential and zero correlation attacks. However, in recent work, Xie et al. reassessed the security of FeW. More precisely, they proved that under the differential fault analysis (DFA) on the encryption states, an attacker can completely recover the master secret key. In this paper, we revisit the block cipher FeW and consider the DFA on its key schedule algorithm, which is rather popular cryptanalysis for kinds of block ciphers. In particular, by respectively injected faults into the 30th and 29th round subkeys, one can recover about $55/80 \approx 69\%$ bits of master key. Then the brute force searching remaining bits, one can obtain the full master secret key. The simulations and experiment results show that our analysis is practical.

Keywords: Differential fault analysis, Block cipher, FeW, Side channel attack

Introduction

Modern cryptographic techniques, including encryption (Wang et al. 2022) and digital signature schemes (Bruinderink and Pessl 2018), are often needed in modern computer networks to guarantee the confidentiality of transmitted messages and authenticity of communication parties. However, many computational efforts caused by the algorithms in cryptographic schemes may be prohibitive for many practical resource-constrained devices (Sadhukhan et al. 2017). For example, in smart transport system, internet of medical things, etc., many devices only have battery life and thus are sensitive to energy consumption. In these cases, the design of cryptographic algorithms should be lightweight. This is also the reason why lightweight cryptography has emerged as a vast research direction. In fact, when the famous Rijndael was selected as Advanced Encryption Standard (AES), there was also a need for lightweight ciphers for specific applications (Zhang et al. 2019). Since then, many lightweight cryptographic algorithms, such as LBlock (Wang et al. 2019), PRESENT (Cnudde and Nikova 2017), GIFT (Xie et al. 2021), Espresso (Bathe et al. 2021), KLEIN (Xiao and Wang 2022), et al., were successively proposed. Inspired by these block ciphers, many researchers intend to design new and more efficient schemes.

FeW cipher, which is a software-oriented design with the aim of achieving high efficiency in software-based environments, was proposed by Kumar et al. (2019). In FeW, the plaintext and ciphertext lengths are both 64 bits, but the key size is 80 or 128 bits. Hence, we call them FeW-80 and FeW-128, respectively. In fact, in FeW, Kumar et al. used a mix Feistel and generalized Feistel structures to enhance its security against several basic cryptographic attacks like differential, linear, impossible differential as well as zero correlation attacks. In addition, the key schedule of FeW is designed in a similar way as PRESENT, which has been chosen as the lightweight encryption standard by International Organisation of Standardisation (ISO).

*Correspondence: changjinyong@xauat.edu.cn

[3] School of Information and Control Engineering, Xi'An University of Architecture and Technology, Xi'An 710055, People's Republic of China
Full list of author information is available at the end of the article

In fact, after designing FeW, Kumar et al. evaluated its security against differential cryptanalysis, impossible differential cryptanalysis, linear cryptanalysis, zero correlation cryptanalysis, and related key cryptanalysis. However, they ignored the differential fault analysis (DFA), which is a quite effective attack to lightweight block ciphers. In fact, DFA is a kind of side-channel attacks, which uses some induced errors to disturb the actual implementation of devices when they are running the encryption or decryption algorithms. The classical ways of inducing faults includes voltage variation, glitch, lazer, etc. (Kim 2012). How to physically induce faults is out of scope of this paper. By analyzing the input–output differentials of S-box as well as the correct/faulty ciphertexts, one can greatly reduce the size of key space. Then the brute force searching can help to find out the true key. Many facts show that DFA is not only effective to block ciphers, but also fatal to many public key cryptographic systems and stream ciphers (Deng and Luo 2021).

In 2020, Xie et al. discussed the DFA security of FeW in the single byte random fault model (Xie et al. 2020). More specifically, they introduced the single byte random fault on the encryption state of FeW and then analyzed the key recovery based on the input–output differentials as well as statistical characteristics of S-box. From their experiment results, one can know that the complete recoveries of master secret keys need an average 47 and 79 fault injections for the two versions of FeW. Moreover, if 2^{10} exhaustive searching is considered for their attack, then the needed numbers of fault injections can be further reduced to 24 and 41, respectively, which shows that the DFA on encryption states of FeW is very effective.

Note that, the encryption algorithm and the key schedule algorithm are both embedded into the hardware devices. Then the random fault injection (of side channel attack) may also occur in the key schedule part. Therefore, considering the DFA on key schedule of block cipher is also necessary. In fact, many important works just discussed this for many famous block ciphers, like AES (Takahashi and Fukunaga 2007), ITUbee (Fu et al. 2017), LBlock (Wei et al. 2018), KLEIN (Gong et al. 2011) et al. Generally speaking, the induced faults can affect the current and subsequent round subkeys, which will further change the encryption states. Hence, the study of DFA on key schedule becomes more complicated than that of encryption state. Then a natural and interesting question arises: If the DFA occurs in the process of key schedule of FeW, then what will happen? In other words, whether FeW is still secure when the attacker launches the fault injections into the key schedule of FeW?

In this paper, we give a positive answer to this question. In particular, for the first time, we consider the DFA on the key schedule of the block cipher FeW. According to the key schedule of FeW, the faulty bits of the previous round subkey will quickly spread to the following keys and encryption states. Thus, we can only analyze the short-key version of FeW: FeW-80, in which we can mix the differential analysis and brute force searching techniques to recover the master secret key. But for Few-128, we do not know how to obtain the master key because the exhaustive searching is still infeasible after the differential analysis. Here, we leave it as an interesting open problem. Hence, we only focus on the discussions about the DFA on the key schedule of FeW-80. Our contributions can be concluded as follows.

- For the first time, we discuss the DFA security of FeW-80 when the random nibble faults are injected into the process of key schedule. In other words, our analysis is in the random nibble model and each fault is induced by half-byte.
- Our differential analysis combines the partial recovery of the master key with brute force searching. More concretely, after injecting faults into the 30th and 29th round subkeys, one can easily recover 55 bits of master secret key. For the remaining 25 bits (of master secret key), we can find out them by exhaustive searching all the possible values.
- Finally, we simulate the encryption, decryption and key schedule processes. The experiment results show that our DFA on FeW's key schedule is practical.

Related works In EUROCRYPT'97, Boneh et al. first considered the faulty analysis on the famous RSA signature (Boneh et al. 1997). At the same time, Biham and Shamir proposed the notion of DFA (Biham and Shamir 1997), which combines fault analysis with differential attack simultaneously, and applied it to DES (Data Encryption Standard). In addition, they also suggested that, in general, the DFA may occur in any process of encryption devices, which naturally includes encryption states as well as the generation of round subkeys. In the following work, Ali et al. discussed the DFA on AES' key schedule algorithm and also showed that their attack is the most efficient (Ali et al. 2013). In 2018, Shibayama et al. analyzed the security of 12 rounds FeW, which has the complexity of 2^{63} (Shibayama et al. 2018). Aayush and Girish also presented the security analysis of FeW based on machine learning approach, which involves using artificial neural network to find the inherited biases present in the design of FeW (Aayush and Girish 2018).

Organizations The rest of this paper is organized as follows. First, we present the symbols and explanations for next parts in Table 1, and in "Review of FeW" section, we review the algorithms of the lightweight cipher FeW. In "Proposed DFA on the key schedule of FeW" section, we

propose our DFA on FeW, which introduces the basic principle of DFA on key schedule, the recoveries of round subkeys and master secret key. In "Simulations and discussions" section, we will give the simulations and discussions on the practical FeW block cipher. Finally, conclusions of this paper are given in "Conclusions" section.

Review of FeW

In this section, we review the construction of FeW, including the algorithms of encryption and decryption as well as key schedule. In FeW, the plaintext length equals to 64-bit, and the ciphertext with the same size is generated based on 80/128 bits key. The total encryption round is 32. The design of FeW is based on the general Feistel structure, which needs fewer computations than SPN structure (used in AES), since only half of the input block is processed through round function in each round.

Encryption algorithm

First, the encryption algorithm parses the 64-bit plaintext P into two 32-bit strings P_0 and P_1. For the right-hand string P_1, it will first become the left-hand one in the next round. Then its XOR with the round key RK_0 will be given to the F-function, whose output is XORed with the left-hand string P_0. The result is set as the right-hand string of the next round. After 32 round transformations, the output of encryption algorithm is just the ciphertext of P, which has the same length (i.e. 64 bits) with P. The overall structure of FeW is presented in Fig. 1.

Table 1 Symbols and explanations

Symbols	Descriptions
DFA	Differential fault analysis
FeW	Feather weight cipher
CPA	Chosen ciphertext attack
MK	80-bit master secret key
MK_i	The i-th intermediate state of MK
K_i	The leftmost 16 bits of MK_i
RK_i	The i-th round subkey
RK_i^j	The j-th nibble of RK_i
C	Correctly generated ciphertext
C^*	The faulty ciphertext
$<<<m$	Left cyclic shift m bits
$[i]_2$	Binary form of i
$\oplus$	Bitwise exclusive-OR operation
$\|\|$	The concatenation of two strings
S	The S-Box of FeW
P	The permutation of FeW
F	The round function
L_1, L_2	The linear functions of FeW
X_i^L	The left part of i-th encryption state
X_i^R	The right part of i-th encryption state
$X_{i,j}^L$	The j-th Nibble of X_i^L
$X_{i,j}^R$	The j-th Nibble of X_i^R

The round function F takes 32-bit string X_i^R and round key RK_i as inputs, and outputs a 32-bit string. That is

$$F : \{0,1\}^{32} \times \{0,1\}^{32} \rightarrow \{0,1\}^{32}.$$

Now, we describe its inner structure in detail. In fact, it includes a permutation P, 8 parallel S-box and two linearly diffuse function L_1 and L_2. The permutation P and S-box S are given as follows (see Table 2 and 3).

The remaining linear functions L_1 and L_2 are from $\{0,1\}^{16}$ to $\{0,1\}^{16}$, and given by the following two equations.

$$\begin{aligned} L_1(x) =& x \oplus (x <<< 1) \oplus (x <<< 5) \oplus (x <<< 9) \\ & \oplus (x <<< 12), \end{aligned}$$

$$\begin{aligned} L_2(x) =& x \oplus (x <<< 4) \oplus (x <<< 7) \oplus (x <<< 11) \\ & \oplus (x <<< 15). \end{aligned}$$

For 32-bit input Y, the calculation of F is presented in Fig. 2.

Decryption algorithm

Since FeW is a balance design with Feistel structure, the decryption algorithm does not need the inversion of round function. In fact, each round subkey is used in the reverse order to obtain the last plaintext. More precisely, for a given 64-bit ciphertext C, which contains two halves C_0 and C_1. Perform the following steps.

1. First apply the round function on the right half C_1 and the round key RK_{31}, and XOR it with C_0 to set the string C_2. That is
$$C_2 = C_0 \oplus F(C_1 \oplus RK_{31}).$$
2. Circularly perform step (1) (using the corresponding round key RK_i) to get the last string C_{32} and C_{33}.
3. Finally, swap the two strings C_{32}, C_{33} to obtain the 64-bit plaintext $P_0||P_1 = C_{33}||C_{32}$.

The correctness of the decryption process can be easily checked.

Key schedule algorithm

This algorithm is used to generate round subkeys for each round from the 80-bit master secret key MK. Parse MK as 80 bits:

$$MK = k_0 k_1 k_2 \ldots k_{78} k_{79}.$$

Then the round subkey RK_i is computed according to the following Algorithm 1.

Algorithm 1. The Generation of Round Subkeys.

Input: 80-bit master secret key MK.

1). Set $MK_0 \leftarrow MK$.

2). **for** $i = 0$ to 63 **do**

MK_i <<<13;

$[k_0k_1k_2k_3] \leftarrow S[k_0k_1k_2k_3]$;

$[k_{64}k_{65}k_{66}k_{67}] \leftarrow S[k_{64}k_{65}k_{66}k_{67}]$;

$[k_{76}k_{77}k_{78}k_{79}] \leftarrow S[k_{76}k_{77}k_{78}k_{79}]$;

$[k_{68}k_{69}k_{70}k_{71}k_{72}k_{73}k_{74}k_{75}] \leftarrow [k_{68}k_{69}k_{70}k_{71}k_{72}k_{73}k_{74}k_{75}] \oplus [i]_2$;

Define K_i as the leftmost 16 bits of MK_i.

End for

2). **for** $i = 0$ to 31 **do**

Set $RK_i \leftarrow K_{2i} \,||\, K_{2i+1}$;

End for

3). Output the round subkeys: RK_0, RK_1,…,RK_{31}.

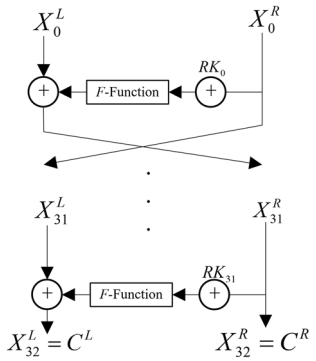

Fig. 1 The overall structure of FeW encryption process

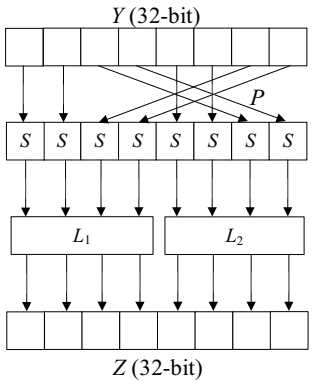

Fig. 2 The *F*-function of FeW

Next, we take the last three round subkeys as examples to show the relationship of their bits. More precisely, assume that

$$MK_{58} = k_0k_1k_2 \ldots k_{79}.$$

Then it naturally holds that

$$K_{58} = k_0k_1k_2 \ldots k_{15},$$

and

$$\begin{aligned} MK_{59} =& S[k_{13}k_{14}k_{15}k_{16}]k_{17}k_{18} \\ & \ldots k_{76}S[k_{77}k_{78}k_{79}k_0]([k_1k_2k_3k_4k_5k_6k_7k_8] \\ & \oplus [59]_2)S[k_9k_{10}k_{11}k_{12}]. \end{aligned}$$

Hence, we have

$$\begin{aligned} RK_{29} =& K_{58}\big|\big|K_{59} = k_0k_1k_2 \ldots k_{15}\big|\big| \\ & S[k_{13}k_{14}k_{15}k_{16}]k_{17}k_{18} \ldots k_{28}. \end{aligned}$$

Table 2 The permutation *P* appeared in encryption process of FeW

x	0	1	2	3	4	5	6	7
P(*x*)	0	1	6	7	4	5	2	3

Table 3 The S-box *S* appeared in encryption process of FeW

x	0	1	2	3	4	5	6	7
$S(x)$	0	1	6	7	4	5	2	3
x	8	9	A	B	C	D	E	F
$S(x)$	B	4	6	B	0	7	3	D

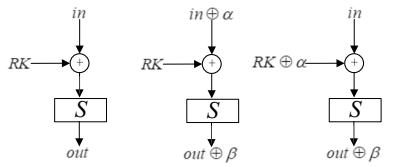

Fig. 3 Difference across S-box of block cipher

Similarly, we can calculate the round subkeys RK_{30} and RK_{31}, which equal to

$$S(k_{26}k_{27}k_{28}k_{29})k_{30}k_{31}\ldots k_{41}||S(k_{39}k_{40}k_{41}k_{42})k_{43}k_{44}\ldots k_{54},$$

and

$$S(k_{52}k_{53}k_{54}k_{55})k_{56}k_{57}\ldots k_{67}||$$
$$S(k_{65}k_{66}k_{67}k_{68})k_{69}k_{70}\ldots k_{76}S(k_{77}k_{78}k_{79}k_0),$$

respectively.

Obviously, the two round subkeys RK_{30} and RK_{31} contain 55 bits of master secret key. Hence, there are at least 25 bits (of master secret key) that cannot be recovered by computing RK_{30} and RK_{31}.

Proposed DFA on the key schedule of FeW

In this section, we describe the DFA on the key schedule of FeW. First, we would like to give the basic principle of DFA on block cipher. Then, present how to apply it to the concrete process of FeW and how to recover the master secret key based on the induced differential faults.

Basic principle of DFA

Since the S-box is the non-linear part of block cipher, the input–output differentials can be used to recover the secret key if the input contains its information. The basic principle is as follows. Let S be an S-box of some block cipher, whose input is the XOR of previous round output and the round key. Figure 3 shows one such example. More concretely, assume that in is the output of previous round and RK is the round subkey. If one fault is injected into the state in or key RK, which causes an input-difference α, then the output of S-box will also have a difference β. Now, if the value $in \oplus RK$ is replaced by X, then we can easily obtain the following equation.

$$S(X \oplus \alpha) \oplus S(X) = \beta. \tag{1}$$

According to the property of S-box of FeW (see Table 3), we know that, for a pair fix and known value (α, β), the above Eq. (1) has 0, 2 or 4 possible solutions for X. Since X is the XOR of in and RK, one can easily get the possible solutions for round key RK if in is known. Finally, other induced faults can be used to find out the true round subkey RK, which is further to be used to recover the master secret key.

Recovery of the last round subkey

In this subsection, we first discuss the recovery of the last round subkey RK_{31} by inducing fault on the generation of RK_{30} in the key schedule algorithm. In fact, we have the following two basic assumptions on the fault model.

Assumption 1 The attacker is allowed to choose any plaintext and get the correct/faulty ciphertexts. That is, this attack is the so-called chosen plaintext attack (CPA).

Assumption 2 The adversary is allowed to induce nibble fault(s) in the process of generating the round keys. But the fault-value induced in some position is unknown.

Now, we first assume that the fault is injected into first nibble of RK_{30}, which causes the differential p. The process of fault propagation is presented in Fig. 4. More specifically, after going through the S-box and L_1-function, the fault will become four "new" ones p_1, p_2, p_3, p_4, which will affect the left half of X_{31}^R. Then the four differentials are reserved to the left half X_{32}^L and affect all the nibbles of the right half X_{32}^R the final ciphertext.

Next, we introduce how to obtain partial round subkey based on these differentials. In fact, this analysis mainly focuses on the final round of encryption process. For the ith S-box S_i, denote by in_i and in_i^* its correct and faulty input, respectively. Then its correct/faulty output (of the S-box) is denoted by out_i/out_i^*. Define

$$out_i \oplus out_i^* = y_{4i}y_{4i+1}y_{4i+2}y_{4i+3}.$$

Moreover, we can easily know that

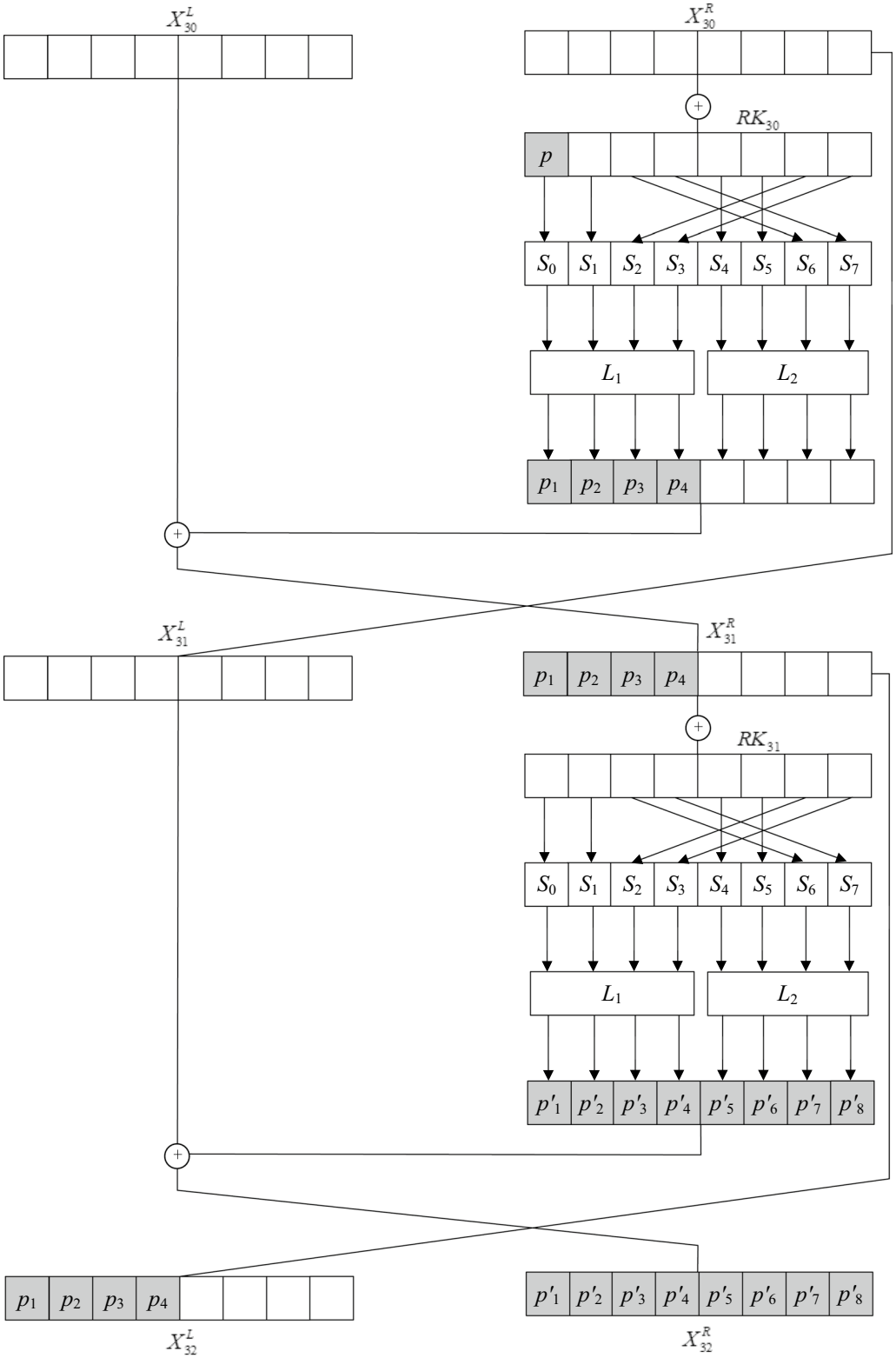

Fig. 4 The fault propagation process on RK_{30}^0

$$p_1 = in_0 \oplus in_0^* = X_{32,0}^L \oplus X_{32,0}^{L,*}, \tag{2}$$

$$p_2 = in_1 \oplus in_1^* = X_{32,1}^L \oplus X_{32,1}^{L,*}. \tag{3}$$

From Fig. 4, we know that

$$\begin{aligned} p_1'||p_2'||p_3'||p_4' &= L_1(out_0||out_1||out_2||out_3) \\ &\quad \oplus L_1(out_0^*||out_1^*||out_2||out_3) \\ &= L_1(out_0 \oplus out_0^*||out_1 \oplus out_1^*||00000000) \\ &= L_1(y_0y_1y_2y_3y_4y_5y_6y_700000000) \\ &= (y_0y_1y_2y_3y_4y_5y_6y_700000000) \\ &\quad \oplus (y_1y_2y_3y_4y_5y_6y_700000000y_0) \\ &\quad \oplus (y_5y_6y_700000000y_0y_1y_2y_3y_4) \\ &\quad \oplus (0000000y_0y_1y_2y_3y_4y_5y_6y_70) \\ &\quad \oplus 0000y_0y_1y_2y_3y_4y_5y_6y_70000) \\ &= (y_0 \oplus y_1 \oplus y_5)||(y_1 \oplus y_2 \oplus y_6)||\cdots||(y_0 \oplus y_4). \end{aligned} \tag{4}$$

For $1 \le i \le 4$, define p_i' as the bit-form: $p_{i,1}'||p_{i,2}'||p_{i,3}'||p_{i,4}'$. Then from Eq. (4), we obtain the following equations.

$$\begin{cases} y_0 = p_{1,4}' \oplus p_{4,3}' \oplus p_{3,4}', \\ y_1 = p_{3,1}' \oplus p_{1,4}' \oplus p_{4,3}' \oplus p_{4,4}' \oplus p_{3,4}', \\ y_2 = p_{4,2}' \oplus p_{3,3}' \oplus p_{4,3}' \oplus p_{4,4}' \oplus p_{3,4}', \\ y_3 = p_{4,3}' \oplus p_{4,4}' \oplus p_{3,4}', \\ y_4 = p_{1,4}' \oplus p_{4,3}' \oplus p_{4,4}' \oplus p_{3,4}', \\ y_5 = p_{3,2}' \oplus p_{4,2}' \oplus p_{3,3}' \oplus p_{4,3}' \oplus p_{4,4}' \oplus p_{3,4}', \\ y_6 = p_{3,3}' \oplus p_{4,3}' \oplus p_{4,4}' \oplus p_{3,4}', \\ y_7 = p_{4,4}' \oplus p_{3,4}'. \end{cases}$$

As a result, we can compute and obtain the eight bits $y_0, y_1, \ldots, y_7$, which are just the bits in $out_0 \oplus out_0^*$ and $out_1 \oplus out_1^*$. Therefore, we can get the output differentials of the two S-boxes S_0 and S_1.

Finally, according to the correspondences of the input–output differentials (i.e. Table 4), we can obtain the candidate values for the inputs in_0 and in_1, which can be used to recover the 1st byte of RK_{31} (i.e. $RK_{31}^0||RK_{31}^1$). Similar analysis on the S-boxes S_6 and S_7 can be made to get the candidate value of the 2nd byte in the round key RK_{31} (i.e.$RK_{31}^2||RK_{31}^3$). The pseudorandom code of recovering the tuple (RK_{31}^0, RK_{31}^1) is given in the following Algorithm 2.

Table 4 The correspondences of input–output differentials for S-Box of FeW

(α, β)	(1, 3)	(1, 7)	(1, A)	(1, C)	(1, D)	(1, E)	(1, F)	(2, 3)	(2, 5)	(2, A)	(2, B)	(2, C)	(2, D)
X	6, 7	C, D	2, 3	0, 1	4, 5	A, B, E, F	8, 9	C, E	4, 6	D, F	1, 3, 5, 7	9, B	0, 2, 8, A
(α, β)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 6)	(3, 8)	(3, D)	(4, 3)	(4, 5)	(4, 6)	(4, B)	(4, E)	(4, F)
X	1, 2	9, A	8, B	D, E	4, 7	5, 6	C, F	9, D	A, E, B, F	2, 6	8, C	0, 4	1, 3, 5, 7
(α, β)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, B)	(5, C)	(6, 3)	(6, 4)	(6, 6)	(6, 8)	(6, 9)	(6, B)	(6, C)
X	1, 4	0, 5	9, C	2, 7	A, B, E, F	3, 6, 8, D	2, 4	3, 5, 7, 8	A, C	1, E	9, F	0, 6	B, D
(α, β)	(7, 1)	(7, 6)	(7, 7)	(7, 8)	(7, 9)	(7, E)	(8, 6)	(8, 7	(8, 9)	(8, A)	(8, C)	(8, D)	(9, 1)
X	A, D	8, F	1, 6, 9, E	0, 7, B, C	3, 4	2, 5	5, D	7, F	0, 2, 8, A	1, 6, 9, E	4, C	3, B	5, C
(α, β)	(9, 3)	(9, 4)	(9, 5)	(9, 6)	(9, 7)	(9, 9)	(9, B)	(A, 1)	(A, 4)	(A, 6)	(A, 9)	(A, C)	(A, D)
X	3, A	6, F	1, 8	0, 9	2, B	7, E	4, D	3, 9	0, 2, 8, A	1, B	6, C	5, F	7, D
(α, β)	(A, F)	(B, 1)	(B, 2)	(B, 8)	(B, A)	(B, B)	(B, E)	(C, 2)	(C, 5)	(C, 7)	(C, 8)	(C, 9)	(C, C)
X	4, E	4, F	5, E	1, A	0, 7, B, C	2, 9	3, 6, 8, D	0, 7, B, C	5, 9	4, 8	3, F	1, D	2, E
(α, β)	(C, F)	(D, 1)	(D, 2)	(D, 5)	(D, 6)	(D, 8)	(D, A)	(D, C)	(D, E)	(E, 1)	(E, 2)	(E, 3)	(E, 9)
X	6, A	6, B	2, F	0, D	3, E	4, 9	5, 8	7, A	1, C	0, E	3, D, 6, 8	1, F	5, B
(α, β)	(E, A)	(E, E)	(E, F)	(F, 1)	(F, 4)	(F, 7)	(F, 8)	(F, D)	(F, F)	(3, 7)	–		
X	4, A	7, 9	2, C	7, 8	4, B	5, A	2, D	1, 6, 9, E	0, F	0, 3			

Algorithm 2. The Pseudorandom Code of Recovering the Tuple (RK_{31}^0, RK_{31}^1).

Input: Correct Ciphertext C and Faulty Ciphertext C^*.

Output: List of the candidates for the tuple (RK_{31}^0, RK_{31}^1).

Parse C as

$$X_{32}^L \| X_{32}^R = \left(X_{32,0}^L \| X_{32,1}^L \| \cdots \| X_{32,7}^L\right) \| \left(X_{32,0}^R \| X_{32,1}^R \| \cdots \| X_{32,7}^R\right),$$

and C^* as

$$X_{32}^{L,*} \| X_{32}^{R,*} = \left(X_{32,0}^{L,*} \| X_{32,1}^{L,*} \| \cdots \| X_{32,7}^{L,*}\right) \| \left(X_{32,0}^{R,*} \| X_{32,1}^{R,*} \| \cdots \| X_{32,7}^{R,*}\right).$$

Compute the differential p_1 from (1) and p_2 from (2).
Compute the differentials

$$p_1' = X_{32,0}^R \oplus X_{32,0}^{R,*},$$
$$p_2' = X_{32,1}^R \oplus X_{32,1}^{R,*},$$
$$p_3' = X_{32,2}^R \oplus X_{32,2}^{R,*},$$
$$p_4' = X_{32,3}^R \oplus X_{32,3}^{R,*}.$$

Solve the system (3) to obtain the 8 bits $y_0, y_1, \cdots, y_7$.

According to the input-output differential table of S_0 and S_1, find the candidate values for in_0 and in_1.

Obtain the candidate values for RK_{31}^0 and RK_{31}^1 by computing

$$RK_{31}^0 = in_0 \oplus X_{32,0}^L,$$
$$RK_{31}^1 = in_1 \oplus X_{32,1}^L.$$

Add the tuple (RK_{31}^0, RK_{31}^1) to the list L_{RK} and return it.

In addition, we can also similarly analyze the process of fault propagation if the fault is injected into other nibbles of the round subkey RK_{30}. In fact, the analysis processes on the nibbles $RK_{30}^0, RK_{30}^1, \ldots, RK_{30}^6$ are same, which are different from that of the last nibble RK_{30}^7. The reason is as follows. If the fault occurs on RK_{30}^7, then it will affect the "correctness" of last round subkey RK_{31}^0. That is, the induced differential p on RK_{30}^7 leads to the new differential p' on RK_{31}^0 (see Fig. 5). As a result, the input differential of the S-box S_0 becomes $p_1 + p'$, which makes the analysis on the L_1 function more difficulty. Nevertheless, it does not affect the analysis on the last two S-boxes S_6, S_7, and thus we can still get the candidate values for the two nibbles RK_{31}^2, RK_{31}^3 by analyzing their input–output differentials.

The above discusses on the recovery of the last round subkey are listed in the following Table 5.

Recovery of the round subkey RK_{30}

In this subsection, we discuss how to further recover the round subkey RK_{30} based on the above recovery of RK_{31}. Now, we need to induce fault into the generation of round subkey RK_{29} in the key schedule. More precisely, for the 8 nibbles in RK_{29}, we consider the differentials induced by the faults and their propagations. A typical case on the fault propagation is described in Fig. 6, in which the fault is induced into the round subkey RK_{29}^1. Here, we remark that the induced fault does not affect the next two round subkeys RK_{30} and RK_{31}.

Now, based on the recovered round subkey RK_{31}, we can decrypt the correct and faulty ciphertexts to obtain the states X_{31}, X_{31}^* after the 31 rounds encryptions. Performing similar analysis as in "Recovery of the last round subkey" section, one can easily get the candidate values of the nibbles $RK_{30}^0, RK_{30}^1, RK_{30}^2, RK_{30}^3$ in RK_{30}. If the faults are induced into other nibbles (of RK_{29}), such as $RK_{29}^2, RK_{29}^3, \ldots, RK_{29}^7$, then the analysis process is also similar.

Next, we further analyze the remaining case. That is, the injected fault occurs in the first nibble of RK_{29}. Still assume that the induced differential is p, which is parsed into $p = b||\overline{p}$. Here, b is its first bit. According to the two possible values of b, we divide our analysis into the following two cases.

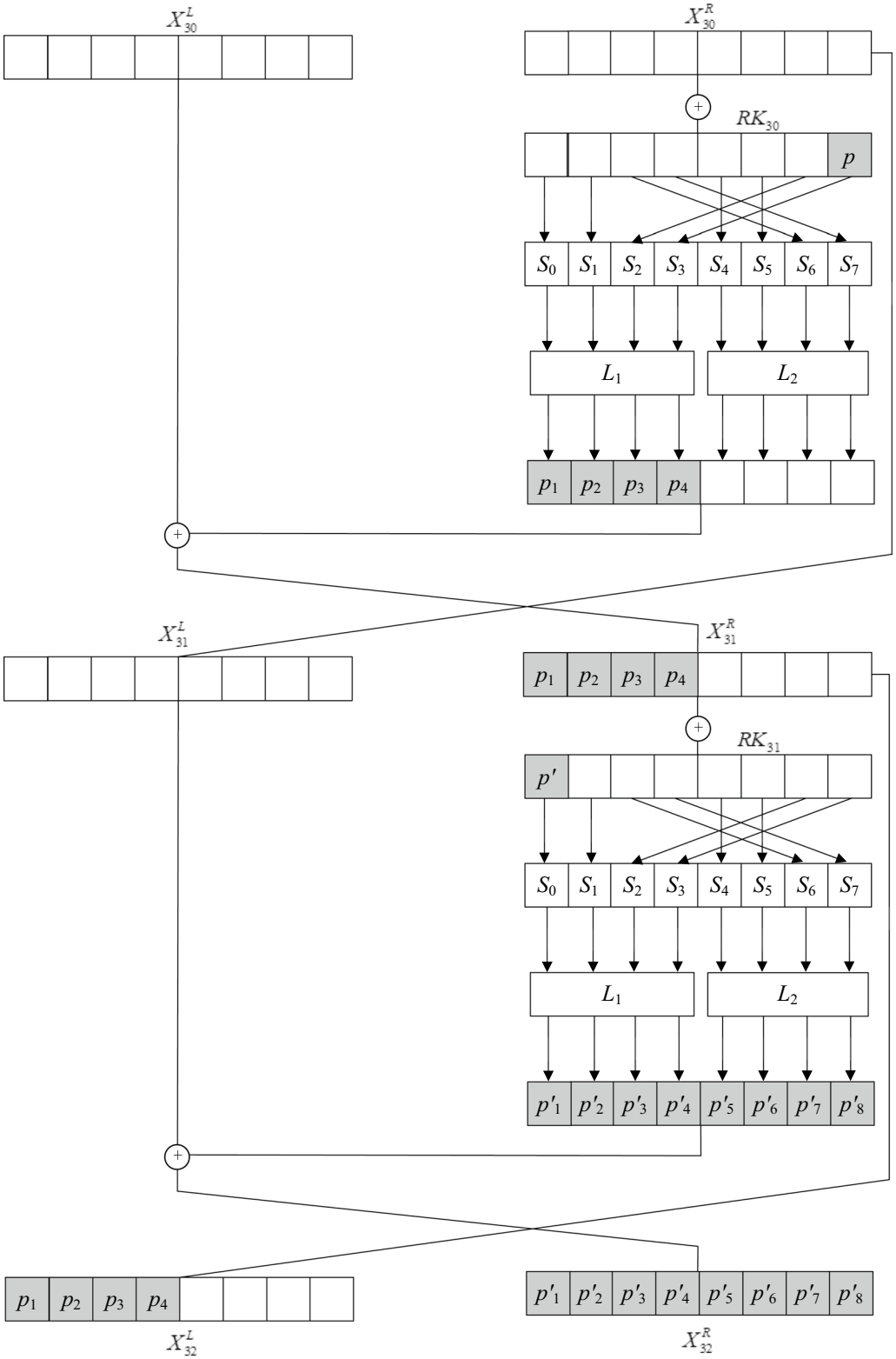

Fig. 5 The fault propagation process on RK_{30}^7

Table 5 The discussions on the recovery of the last round subkey

Fault on RK_{30}	RK_{30}^0	RK_{30}^1	RK_{30}^2	RK_{30}^3
Recovery of RK_{31}	RK_{31}^0, RK_{31}^1	RK_{31}^0, RK_{31}^1	RK_{31}^4, RK_{31}^5	RK_{31}^4, RK_{31}^5
	RK_{31}^2, RK_{31}^3	RK_{31}^2, RK_{31}^3	RK_{31}^6, RK_{31}^7	RK_{31}^6, RK_{31}^7
Fault on RK_{30}	RK_{30}^4	RK_{30}^5	RK_{30}^6	RK_{30}^7
Recovery of RK_{31}	RK_{31}^4, RK_{31}^5	RK_{31}^4, RK_{31}^5	RK_{31}^0, RK_{31}^1	RK_{31}^2, RK_{31}^3
	RK_{31}^6, RK_{31}^7	RK_{31}^6, RK_{31}^7	RK_{31}^2, RK_{31}^3	

Case 0. If $b=0$, then decrypt the final ciphertexts with the help of the recovered last round subkey RK_{31}.
Case 1. If $b=1$, then define the last nibble of RK_{31} as $S(k_{77}k_{78}k_{79}(k_0 \oplus 1))$ and decrypt the final ciphertexts by using the updated round subkey RK_{31}.

The decrypted states (under the round subkey RK_{31}) will be used to further analyze the recovery of the round subkey RK_{30} by using previous steps.

Finally, we explain why our above analysis will give correct recovery of the round subkey RK_{30}. According to the key schedule of FeW, the last nibble of RK_{31} equals to $S(k_{77}k_{78}k_{79}k_0)$ if the first nibble of RK_{29} is $k_0k_1k_2k_3$. If $b=0$, then the induced differential p does not affect the last round subkey RK_{31}, which can be used to correctly decrypt the final ciphertexts to obtain the states of the 31 round encryptions. If $b=1$, then the induce differential p changes the bit k_0 into $k_0 \oplus 1$, which will also appear in the last round subkey RK_{31}. Thus, the last round subkey, which will be used to recover the inner states of encryption process, needs to be updated according to the induced differential.

The above discusses on the recovery of the round subkey RK_{30} are listed in the following Table 6.

Recovery of the master key

According to the key schedule, we can easily know that the master key can be recovered from any intermediate key state MK_i, which consists of 29 bits of RK_{31}, 26 bits of RK_{30}, and 25 bits of RK_{29}. If we can recover all the bits of RK_{31} and RK_{30} from the above discussions, then the remaining 25 bits can be obtained by brute force searching, whose time complexity is 2^{25}. In addition, we remark that, in the process of recovering the round subkey RK_{30}, the two cases on the first bit of induced differential p increases the total complexity from 2^{25} to 2^{26}. Therefore, by inducing several nibble faults into the key schedule of FeW, one can finally recover the master key based on the differential analysis on RK_{31} and RK_{30}, as well as the brute force searching. The total time complexity equals to 2^{26}, which is much lower than the original 2^{80}.

Simulations and discussions

In this section, we present the simulations and discussions on the differential fault attack on FeW. Specifically, we will choose a fix plaintext "0×1234567890ABCDEF" and several randomly master secret keys, and compute the ciphertexts under these keys. Then by simulating the fault-injections into the 29th and 30th round subkeys, we get the corresponding faulty ciphertexts. Based on the correct and faulty ciphertext, one can recover the round subkeys RK_{31} and RK_{30}, respectively. Finally, the remaining unknown bits of master secret key is found out by brute force searching. The concrete simulation process is given as follows.

Step 1. Randomly choose a master secret key and compute the round subkeys according to key schedule.
Step 2. Generate the correct ciphertexts under the master secret key.
Step 3. Simulate the fault-injection into RK_{30} and obtain faulty ciphertexts.
Step 4. Search the candidate values for the nibbles of RK_{31}.
Step 5. Simulate the fault-injection into RK_{29} and obtain faulty ciphertexts.
Step 6. Search the candidate values for the nibbles of RK_{30}.
Step 7. Exhaustive search the remaining bits (of MK_{29}) by re-encrypting and decrypting.
Step 8. Compute the master secret key by reversing the key schedule of FeW.

Simulations

The whole simulation of FeW, including the encryption and decryption algorithms, is in the C programming language and based on a desktop with the configuration of Intel(R) Core (TM) i5-10210U CPU @1.60 GHz and 16 GB RAM. We simulate the DFA process 10 times to get significant data. Since their final results are similar, we only show five typical examples in the following Table 7, which presents the number of needed faults and their running time of recovering master secret key. Here, we remark that the factors including the positions of induced faults, the input–output differentials of S-box, and the particular master secret key, can affect the simulation results. Therefore, in each simulation, the needed number of faults may be different from other ones.

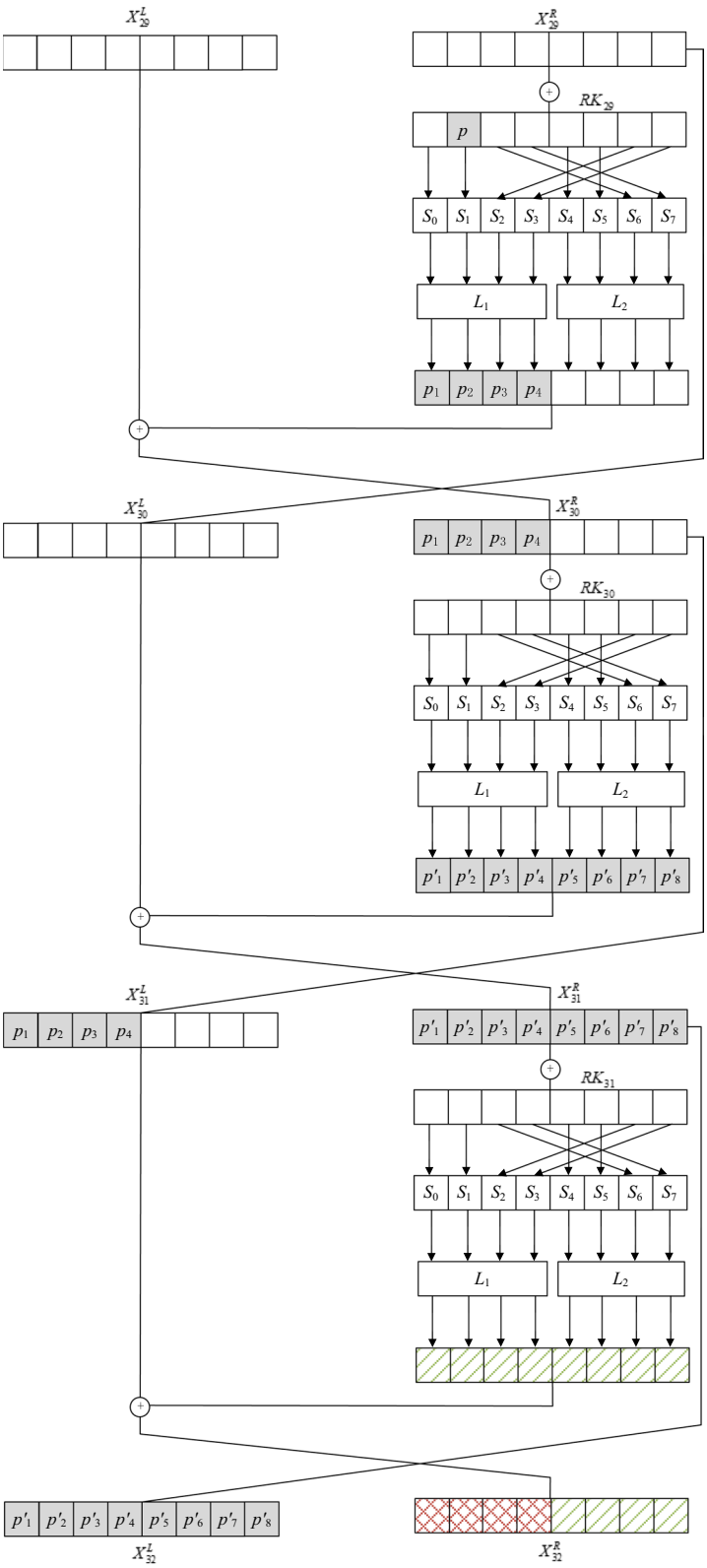

Fig. 6 The fault propagation process on RK_{29}^1

Table 6 The discussions on the recovery of round subkey RK_{30}

Fault on RK_{29}	RK_{29}^0	RK_{29}^1	RK_{29}^2	RK_{29}^3
Recovery of RK_{30}	RK_{30}^0, RK_{30}^1	RK_{30}^0, RK_{30}^1	RK_{30}^4, RK_{30}^5	RK_{30}^4, RK_{30}^5
	RK_{30}^2, RK_{30}^3	RK_{30}^2, RK_{30}^3	RK_{30}^6, RK_{30}^7	RK_{30}^6, RK_{30}^7
Fault on RK_{29}	RK_{29}^4	RK_{29}^5	RK_{29}^6	RK_{29}^7
Recovery of RK_{30}	RK_{30}^4, RK_{30}^5	RK_{30}^4, RK_{30}^5	RK_{30}^0, RK_{30}^1	RK_{30}^2, RK_{30}^3
	RK_{30}^6, RK_{30}^7	RK_{30}^6, RK_{30}^7	RK_{30}^2, RK_{30}^3	

From the above simulation, one can easily know that our proposed differential fault attack, which reduces the key space from 2^{80} to 2^{26}, is a rather practical attack for the block cipher FeW. In average, the recoveries of the two round subkeys RK_{31} and RK_{30} need 12.6 induced faults, and the exhaustive searching time for the remaining bits needs 5.3 min.

Finally, we remark that the value 2^{26} is the lower bound of our proposed DFA on FeW. We explain it as follows. According to our attack, the two models of induced faults can only be used to recover the last two round subkeys, which contain 55 bits of master secret key, although the faults also affect the round subkey RK_{29}. But we do not know how to recover the remaining bits based on the existing differentials. A natural idea is to induced additional faults into the previous round subkey RK_{28}, which may descend the lower bound by recovering additional bits of RK_{29}. However, this will make the analysis much complicated. The reason is as follows. The induced differentials on RK_{28} will also affect the bits of RK_{31}. But the quantitative differentials on RK_{31} cannot be calculated. As a result, only from correct/faulty ciphertexts, one cannot decrypt them with the help of "faulty" round subkey RK_{31}.

Discussions

In this subsection, we present the comparison with Xie et al.'s work (Xie et al. 2020) since it also consider the DFA on the same FeW algorithm. As we discussed in Introduction, the main difference between them lies in that Xie et al.'s work induces single byte random faults on the encryption state while our paper considers the random nibble faults on the key schedule. The concrete comparisons are listed in the following Table 8.

Here, we remark that this table only gives the average numbers of needed faults for both works.

Conclusions

Differential fault analysis is a popular side channel attack to block cipher. In this paper, we apply the DFA to the lightweight block cipher FeW. More specifically, in the nibble model, we consider the DFA to the key schedule of FeW-80. By inducing faults into the 30th and 29th round subkeys, and analyzing the input–output differentials of S-box, one can easily obtain the candidate values for the nibbles of RK_{31} and RK_{30}. Then brute force searching the remaining bits can finally recover the original master secret key. However, it seems that this technique only works for 80 bits FeW. For FeW-128, the only recoveries of last round subkeys seem to be not enough because its key space is too large. Thus, it may be an interesting work to investigate the DFA on key schedule of FeW-128. Finally, the proposed simulations show that our proposed attack is rather practical for FeW-80.

Table 7 Five typical examples

Random 80 bits master secret key	**28b073e9 d9d55414**	**116394d0 d827379d**	**d3f2a25 f9084a2e**	**a618c845 62b40c9c**	**9f13f86d 24cbdeb3**
Number of faults on RK_{30}	5	7	8	5	6
Number of faults on RK_{29}	7	6	6	6	7
Time of exhaustive searching (Min)	5.21	5.33	5.42	5.17	5.36

Table 8 The comparisons with Xie et al.'s work

	Position of faults	**Fault model**	**Round(s) of fault injection**	**Number of needed faults**
Xie et al.'s work	Encryption state	Random byte fault	29, 30, 31	24.9 with 2^{10} exhaustive searching
Our work	Key schedule	Random nibble fault	29, 30	12.6 with 2^{26} exhaustive searching

Acknowledgements
The authors would like thank the anonymous reviewers for their invaluable suggestions and comments.

Author contributions
HX and LW proposed the DFA on FeW and implemented the algorithms proposed in this paper. JC participated in problem discussions and improvements of the manuscript. All authors read and approved the final manuscript.

Funding
This work is supported in part by the Foundation of State Key Laboratory of Information Security under Grant 2021-MS-04, and in part by the Natural Science Foundation of Shaanxi Province under grant 2022-JM-365.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Ethics approval was not required for this research.

Competing interests
Authors declare that they have no competing interests.

Author details
[1]School of Computer Science, Northwestern Polytechnical University, Xi'An 710055, People's Republic of China. [2]Engineering University of PAP, Xi'An 710055, People's Republic of China. [3]School of Information and Control Engineering, Xi'An University of Architecture and Technology, Xi'An 710055, People's Republic of China. [4]State Key Laboratory of Information Security (SKLOIS), Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100089, People's Republic of China.

Received: 8 June 2022 Accepted: 7 August 2022
Published online: 03 November 2022

References

Aayush J, Girish M (2018) Analysis of lightweight block cipher FeW on the basis of neural network. In: Harmony search and nature inspired optimization algorithms. Springer, Berlin, pp 1041–1047
Ali S, Mukhopadhyay D, Tunstall M (2013) Differential fault analysis of AES: towards reaching its limits. J Cryptogr Eng 3:73–97
Bathe B, Tiwari S, Anand R, et al (2021) Differential fault attack on Espresso. In: INDOCRYPT'21. Springer, Berlin, pp 271–286
Biham E, Shamir A (1997) Differential fault analysis of secret key cryptosystem. In: Proceedings of CRYPTO, pp 513–525
Boneh D, Demillo R, Lipton R (1997) On the importance of checking cryptographic protocols for faults. In: EUROCRYP'97. Springer, Berlin, pp 37–51
Bruinderink L, Pessl P (2018) Differential fault attacks on deterministic lattice signatures. In: eprint IACR'2018, vol 335, pp 1–25
Cnudde T, Nikova S (2017) Securing the PRESENT block cipher against combined side-channel analysis and fault attack. IEEE Trans Very Large Scale Integrat Syst 25:3291–3301
Deng Y, Luo H (2021) A distributed identity authentication scheme for differential fault attack. In: ICCT'21. IEEE, pp 731–735
Fu S, Xu G, Pan J, Wang Z, Wang A (2017) Differential fault attack on ITUbee block cipher. ACM Trans Embedded Comput Syst 16(2):1–10
Gong Z, Nikova S, Law Y (2011) KLEIN: a new family of lightweight block ciphers. In: RFIDSec'11. Amherst, USA, pp 1–18
Kim C (2012) Improved differential fault analysis on AES key schedule. IEEE Trans Inf Forensics Secur 7(1):41–50
Kumar M, Pal S, Panigrahi A (2019) FeW: a lightweight block cipher. Turk J Math Comput Sci 11(2):73–58
Sadhukhan R, Patranabis S, Ghoshal A et al (2017) An evaluation of lightweight block ciphers for resource-constrained applications: area, performance, and security. J Hardw Syst Secur 4:1–16
Shibayama N, Igarashi Y, Kaneko T (2018) A new higher order differential of FeW. In: CANDARW, LoS Alamitos, pp 466–471
Takahashi J, Fukunaga T (2007) DFA mechanism on the AES key schedule. In: FDTC'07. IEEE Computer Society, pp 62–74
Wang H, Feng L, Ji Y, Shao B, Xue R (2022) Toward usable cloud storage auditing, revisited. IEEE Syst J 16(1):693–700
Wang T, Wang Y, Gao Y et al (2019) Differential fault attack on lightweight block cipher LBlock. J Cryptol Res 6(1):18–26
Wei Y, Rong Y, Fan C (2018) Differential fault attacks on lightweight cipher LBlock. Fundam Inform 157(1–2):125–139
Xiao H, Wang L (2022) The differential fault analysis on block cipher KLEIN-96. J Inf Secur Appl 67:103205
Xie M, Li J, Tian F (2020) Differential fault attack on FeW. J Commun 41(4):143–149
Xie M, Tian F, Li J (2021) Differential fault attack on GIFT. Chin J Electron 30(4):669–675
Zhang J, Wu N, Li J (2019) A novel differential fault analysis using two bytes fault model on AES key schedule. IET Circut Dev Syst 13(5):661–666

Publisher's Note