Efficient Differential Fault Analysis for AES

Shigeto Gomisawa[1], Yang Li[1], Junko Takahashi[1,2], Toshinori Fukunaga[2], Yu Sasaki[2], Kazuo Sakiyama[1], Kazuo Ohta[1]

[1] Department of Informatics, The University of Electro-Communications
1-5-1 Chofugaoka, Chofu, Tokyo 182-8585, Japan
g-shigeto-lfat@ice.uec.ac.jp
[2] NTT Information Sharing Platform Laboratories, NTT Corporation
3-9-1 Midori-cho, Musashino-shi, Tokyo 180-8585, Japan

Abstract. This paper proposes improved post analysis methods for Differential Fault Analysis (DFA) against AES. In detail, we propose three techniques to improve the attack efficiency as 1) combining previous DFA methods, 2) performing a divide-and-conquer attack by considering the AES key-schedule structure, and 3) taking the linearity of the MixColumns operation into account. As a result, the expectation of the analysis time in the previous work can be reduced to about one sixteenth. Notice that these improvements are based on the detailed analysis of the previous DFA methods and the calculation time and memory cost in practical implementations. Moreover, the proposed techniques can be widely applied to DFA attacks under different assumptions.

Keywords: Fault Analysis Attack, DFA, AES, Divide-and-Conquer

1 Introduction

Fault Analysis (FA) attacks retrieve secret information by analyzing a faulty ciphertext that is obtained by inducing errors into a cryptographic device while it is working. In 1997, Biham and Shamir applied FA to Data Encryption Standard (DES) [1]. The method is called Differential Fault Analysis (DFA). Since then, several DFA methods for AES have been proposed in [2–8].

Many of previous methods are based on [5] which uses a 1-byte fault at the *random fault model*, where an attacker does not know the position and value of the fault. The attack procedure of [5] and [7] is summarized in Alg.1.

Algorithm 1. Attack algorithm by [5] and [7]

Step 0. Guess the position of the fault injected byte.
Step 1. Reduce the key-candidate space of the last round key K^{10} into $(2^8)^4 = 2^{32}$ by using a pair of the correct and faulty ciphertexts.
Step 2. 2^{32} candidates of K^{10}, retrieval the original key K by the brute force attack. (10 AES-round operations per candidate)

If the correct key is not found, the attack is repeated with changing the guess at Step 0. We see that the number of repetitions of Step 0 in [5] is 13 at the worst case. In 2009, Saha, Mukhopadhyay and RoyChowdhury [7] showed an improvement of [5] so that the number of repetitions could be at most 4 instead of 13. Tunstall and Mukhopadhyay [8] showed another improvement of [5] so that the Step 2 can be divided into 2^{32} times with a 2-round AES decryption and 2^8 times with a 10-round AES decryption.

In AES, due to the mixture of ShiftRows and MixColumns, the full diffusion reaches after a 2-round operation. Therefore, Step 2 of [5] and [7], which computes a 10-round decryption, and Step 2 of [8], which first computes a 2-round decryption, require the knowledge of all bytes of K^{10}. This fact reveals that all previous DFA methods need a 2^{32} brute force attack which costs the most time.

In this research, we propose several improved DFA methods based on [5]. First, we point out that the techniques of [7] and [8] can be combined in order to reduce the analysis time. Second, we present a divide-and-conquer attack at Step 2 with considering the AES key-schedule structure, which significantly reduce the space of the key candidates. Previous methods retrieve the correct key by computing the input difference of the 8^{th}-round MixColumns from a pair of a correct and a faulty ciphertext, which requires the inverse operations of ShiftRows and MixColumns twice for each. In this case, the diffusion reaches the full diffusion.

Meanwhile, we retrieve the correct key by pre-computing the possible output differences (4-byte) of MixColumns in the 8^{th} round and using only 2 bytes of them in Step 2. Hence, Step 2 can compute the inverse of ShiftRows and MixColumns twice and once, respectively, and thus the diffusion does not reach the full diffusion. By taking advantage of this, we could perform Step 2 with the partial knowledge of K^{10}, and thus a divide-and-conquer attack can be applied potentially. As a result, the complexity of Step 2 is reduced to 2^{30} 1-round decryptions.

Lastly, we further improve this result by considering the linearity of MixColumns in the 9^{th} round and apply a divide-and-conquer attack. As a result, the complexity of Step 2 is reduced to 2^{28} 1-round decryption, and the number of repetitions at Step 0 is at most 4.

1.1 Notations

The following notations are used in this paper. Notations of the intermediate states of AES are following [9].

2 Previous DFA methods for AES

2.1 Overview

In 2003, Piret and Quisquater proposed a DFA method for 128-bit AES (AES-128) [5]. They indicated that an attacker retrieves a secret key with a pair of a correct ciphertext C and a faulty ciphertext $\overline{C}$ obtained by injecting 1-byte error

C, $\overline{C}$: correct and faulty ciphertexts, respectively
K^n : n^{th} round key
i, j : row and column positions of the state
SB, *SR*, *MC*: *SubBytes*, *ShiftRows*, and *MixColumns* functions
I^n : input of the n^{th} round
SBO^n, MCI^n : output of *SB* and input of *MC* in the n^{th} round
MCO^n : output of *MC* in the n^{th} round$X$
$C_{i,j}$, $\overline{C_{i,j}}$, $K^n_{i,j}$, $I^n_{i,j}$: (i,j) byte of C, $\overline{C}$, K^n, and I^n
t_X : computational time for an operation X

into I^8. Because an attacker does not know the error byte position, the attack needs to be repeated for 13 times at the worst case.

Saha, Mukhopadhyay and RoyChowdhury improved this method by utilizing the nature of the AES structure that a 1-byte error injected at I^8 is diffused to 4 bytes by MC in the 8^{th} round, and to 16 bytes by MC in the 9^{th} round. The error path varies by positions where the 1-byte error is injected. Figure 1 describes all of the error paths when a 1-byte error is injected into I^8. The numbers "1" to "4" in Fig. 1 represent the propagation paths of the error. As shown in the figure, we see that the number of the paths is only 4. Consequently, Saha *et al.*'s method indicates that an attacker can recover a secret key by repeating the attack at most 4 times.

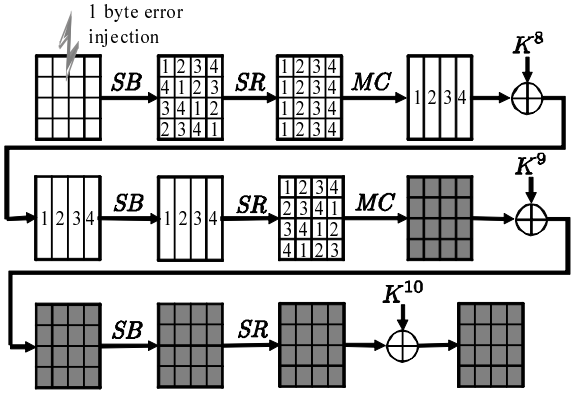

Fig. 1. Error propagating paths when 1 byte error is injected at I^8

Tunstall and Mukhopadhyay also proposed an efficient DFA method based on the one proposed by Piret and Quisquater. It assumes that an attacker knows in which byte an error is injected at I^8. Using the inverse key expansion routine, they divide Step 2 into two phases as shown in Alg. 2.

Algorithm 2. Improved procedure for Step 2 in [8]

Step 2-1. For 2^{32} candidates of K^{10}, compute the 2-round decryption for C and $\overline{C}$. The correct key has a non-zero difference in the guessed byte position at Step 0 and zero difference for other 3 bytes. This occurs with a probability of 2^{-24}. Hence only 2^8 candidates remain after this step.

Step 2-2. Recover a secret key by the brute force attack for remaining 2^8 candidates, which are reduced at Step 2-1.

Table 1. Comparison of the analysis time of previous DFA methods

		Piret *et al.* [5]	Saha *et al.* [7]	Tunstall *et al.* [8]
Step 1		$3 \cdot 2^{16}(2 \cdot t\_AR + t\_CMP\_2^8) \cdot 4$		
Step 2-1A		——————-		
Step 2-1B		——————-		$V_2 = 2^{32} \cdot (t\_CMP + 2 \cdot 2 \cdot t\_AR + t\_KR)$
Step 2-2		$V_1 = 2^{32}(t\_CMP + 10 \cdot t\_AR + 10 \cdot t\_KR)$		$2^8 \cdot (t\_CMP + 10 \cdot t\_AR + 10 \cdot t\_KR)$
Number of trials	best	1		
	worst	13	4	16
	E	3.4	2.5	8.5
Total		$3.4 \cdot V_1$	$2.5 \cdot V_1$	$8.5 \cdot V_2$
Ratio($\alpha = \beta = 1$)		100	74	71
Ratio($\alpha = \beta = 0$)		100	74	100
Memory[Byte]		$4 \cdot 2^{12}$		

t_AR represents the processing time of computing one round of AES. t_KR represents the processing time of computing one round of key-schedule. $t\_KR = \beta \cdot t\_AR (0 \leq \beta \leq 1)$, t_CMP represents the processing time of comparing two data. $t\_CMP\_2^x = x \cdot t\_CMP$, $t\_CMP = \alpha \cdot t\_AR (0 \leq \alpha \leq 1)$ We measure the attack efficiency ratio by setting the method by Piret and Quisquater "100".

Both methods [5, 7] do not have Step 2-1, while the method in [8] does. Because Step 2-1 performs only a 2-round decryption, it is more efficient than the original Step 2 which performs a full-round (10-round) AES decryption. Also since the key space is largely reduced at Step 2-1 before the full-round decryption, the method by Tunstall and Mukhopadhyay can be more efficient. However their attack needs to be repeated 16 times at the worst case.

2.2 The number of trails in previous DFA methods

We consider the expectation of the number of repetitions of Alg. 1 (Piret *et al.*'s method). In this attack, an attacker guesses the position of the injected error propagation byte, and generates the error propagation path based on the guess. The attack succeeds if the generated error propagation path is the same as the actual error propagation path. Hence, the expectation of the number of trials (E) of the Piret *et al.*'s method can be calculated by Eqs. (1) and (2). Therefore, E

is 3.4. Note that p_n is the probability that the first success of the attack occurs at the n^{th} trial.

$$p_n = \begin{cases} \frac{1}{4} & (n = 1) \\ \frac{4}{16-(n-1)} \times \prod_{k=1}^{n-1}(1 - p_k) & (n > 1), \end{cases} \quad (1)$$

$$E = \sum_{n=1}^{13}(n \cdot p_n). \quad (2)$$

For the case of Saha *et al.*'s method, the number of trials is 1 at the best case, and 4 at the worst case. Consequently, the expectation E is $\frac{(1+4)}{2}$ or 2.5. Finally, we consider the case of Tunstall *et al.*'s method. The number of trials is 1 at the best case, but 16 at the worst case. Therefore, the expectation is $\frac{(1+16)}{2}$ or 8.5. Table 1 compares the computational complexity of previous DFA methods. The expectation of the number of trials is also compared.

Evaluate the analysis time We evaluate the method by Piret and Quisquater. Before performing Alg. 1, they pre-compute the difference table of MCO^9. The difference table of MCO^9 has 2^8 entries. Step 1 reduces the space of the key candidates of the last round key K^{10} per column of the AES state. Instead of checking 4 bytes in a column at once, we can perform 2-byte checking for 3 times for a efficient computation. The details were explained in [5]. Consequently, the analysis time of Step 1 is considered as $3 \cdot 2^{16}(2 \cdot t\_AR + t\_CMP\_2^8) \cdot 4$. Hereafter, we denote the comparison with a table of 2^x entries as CMP_2^x.

Step 2 is a step where an attacker performs 2^{32} brute force attack. Hence, the analysis time of Step 2 is $V_1 = (t\_CMP + 10 \cdot t\_AR + 10 \cdot t\_KR) \cdot 2^{32}$. Consequently, the analysis time of Step 1 is negligibly small compared with Step 2. By considering the expectation of the number of trials, the analysis time is $3.4 \cdot V_1$.

Next, we consider the analysis time of the method by Saha, Mukhopadhyay and RoyChowdhury. Their method is almost the same as the method by Piret and Quisquater, while the expectation of the number of trials E is only 2.5. Consequently, the analysis time of their method is derived as $2.5 \cdot V_1$.

We consider the analysis time of the method by Tunstall and Mukhopadhyay. The complexity of Step 1 is calculated based on the attack procedure of [5]. Because Step 1 is not related to our improvement and its computational complexity is small enough, we will omit the detailed explanation. For Step 2-1, an attacker performs a 2-round decryption for 2^{32} candidates of K^{10}. Since K^9 needs to be calculated from K^{10} in a 2-round decryption, we perform one AES key-schedule operation denoted as KR. Next, we check whether the derived differential values by the 2-round decryption contains only a 1-byte error. We call this operation CMP.

Hence, the analysis time of Step 2-1 is $V_2 = (t\_CMP + 2 \cdot 2 \cdot \_AR + t\_KR) \cdot 2^{32}$. Note that the number of remaining candidates becomes 2^8 after Step 2-1. For

Step 2-2, an attacker performs a 10-round decryption for 2^8 candidates, and thus the complexity becomes $(t\_CMP + 10 \cdot t\_AR + 10 \cdot t\_KR)) \cdot 2^8$. Because the analysis times of Steps 1 and 2-2 are negligibly-small compared with Step 2-1, and as a result, the analysis time of their method is $8.5 \cdot V_2$.

In order to compare the analysis time of previous methods, we measure the ratio by setting that the computational complexity of the method by Piret and Quisquater is 100. We assume that the processing times of CMP and KR are represented with t_AR as follow.

1. An attacker sorts 2^8 difference table while preparing the table and uses a binary search when performing CMP, and hence we can assume that $t\_CMP\_2^x = x \cdot t\_CMP$
2. We assume $t\_CMP = \alpha \cdot t\_AR$, where $0 \leq \alpha \leq 1$.
3. We assume $t\_KR = \beta \cdot t\_AR$, where $0 \leq \beta \leq 1$.

In a practical implementation of the DFA method, the above assumptions are dependent on computing systems. However, we believe that AR is the most time consuming operation, so we set $0 \leq \alpha, \beta \leq 1$.

For the purpose of evaluating the impact of the computational complexity on different platforms (e.g. PC with AES-NI and dedicated hardware implementations)

When $\alpha = \beta = 1$, we obtain $V_1 = (10 + 10 + 1) \cdot t\_AR \cdot 2^{32} = 20 \cdot 2^{32} \cdot t\_AR$ and $V_2 = (1+4+1) \cdot 2^{32} = 6 \cdot 2^{32} \cdot t\_AR$. In this condition, the methods of [7] and [8] achieve 74 and 71, respectively in terms of *Ratio* in Table 1. On the contrary, for the case of $\alpha = \beta = 0$, we obtain $V_1 = 10 \cdot t\_AR \cdot 2^{32} = 10 \cdot 2^{32} \cdot t\_AR$ and $V_2 = (0 + 4 + 0) \cdot 2^{32} = 4 \cdot 2^{32} \cdot t\_AR$. In this condition, the methods of [7] and [8] achieve 74 and 100, respectively in terms of *Ratio*.

Note that all of the three methods use at most $4 \cdot 2^{12}$-byte memory during performing the DFA attacks.

3 Our proposed DFA methods

3.1 Combining previous DFA methods

Basic idea for the combined method Saha *et al.* reduce the expectation of the number of trials, while Tunstall *et al.* reduce the analysis time of Step 2. These two improvements use different properties of the attack, and therefore it is natural to combine them for achieving an efficient DFA.

In the combined method, an attacker guesses the error-propagation path. Therefore, when Step 2-1 is computed, an attacker checks whether the difference of MCI^8 contains non-zero in any 1 byte and zeros for other 15 bytes. This occurs with a probability of $4 \times 2^{-24} = 2^{-22}$, and hence $2^{32} \times 2^{-22} = 2^{10}$ key candidates remain. Finally, we perform a brute force attack with 2^{10} candidates in Step 2-2. Consequently, we can see that the analysis time is reduced to $2.5 \cdot V_2$

where V_2 will be explained in Sect. 3.1.2. The details of the DFA efficiency will be discussed in the next sub-section. Note that this combining idea is also applicable for the attacks in the following sections.

Computational complexity of the combined method Step 1 of this method is the same as one in the methods by [7], [8], which cannot be a critical operation since its computational complexity is at most $120 \cdot 2^{16}$ or about 2^{23} considering the case of $\alpha = 1$. Instead, Step 2-1B is the most time-consuming operation as is the case for the method in [8]. As the number of trials is the same as that of the method in [7], the expectation of the total computational complexity is $2.5 \cdot V_2$ where V_2 is the computational complexity in Step 2-1B. As a result, we see that the reduction of $\frac{2.5}{8.5}$ or 29% is possible compared to the method in [8] as shown in Tables 1 and 2.

3.2 Divide-and-Conquer by considering the key schedule

In this section, we apply two kinds of DFA improvements. First, we reduce the computation in Step 2-1 of Alg. 1 by introducing a look-up table that keeps the values for I^{10} when performing Step 1. Then, we show a divide-and-conquer attack by considering the structure of the AES key schedule.

Improving Step 2-1 of Alg. 2 Before introducing how to reduce a half of the calculation time for the Step 2-1 of Alg. 2, we review some details of Step 1. Note that the position of the fault-injected byte is fixed in Step 1. The attacker first computes the output difference of SB in the 10^{th} round (ΔSBO^{10}) from C and $\overline{C}$. Then, based on the guesses of the differential value for fault-injected byte

Table 2. Comparison of the analysis time of our improved DFA methods

		Ours (Section 3.1)	Ours(Section 3.2)	Ours (Section 3.3)
Step 1		$3 \cdot 2^{16}(2 \cdot t\_AR + t\_CMP\_2^8) \cdot 4$		
Step 2-1A		———-	$V_3 = 2^{30}(2 \cdot t\_AR + t\_KR + t\_CMP\_2^{10})$	$V_4 = 2^{28}(2 \cdot t\_AR + t\_KR + t\_CMP\_2^{20})$
Step 2-1B		$V_2 = 2^{32}(t\_CMP + 2 \cdot 2 \cdot t\_AR + t\_KR)$	$V_5 = 2^{26}(t\_CMP + 2 \cdot 2 \cdot t\_AR + t\_KR)$	
Step 2-2		$2^{10}(t\_CMP + 10 \cdot t\_AR + 10 \cdot t\_KR)$		
Number of trials	best	1		
	worst	4		
	E	2.5		
Total		$2.5 \cdot V_2$	$2.5 \cdot V_3$	$2.5 \cdot (V_4 + V_5)$
Ratio($\alpha = \beta = 1$)		21	11	5.4
Ratio($\alpha = \beta = 0$)		29	3.7	1.4
Memory[Byte]		$4 \cdot 2^{12}$	$4 \cdot 2 \cdot 2^{12}$	2^{20}

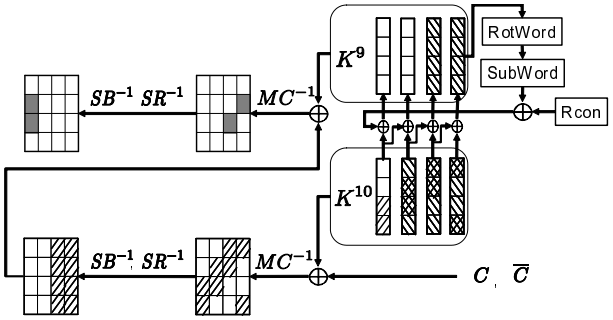

Fig. 2. The last two rounds of the key schedule and the last round of the encryption in AES-128

in MCI^9, the corresponding 4-byte difference of I^{10} can be computed. Based on the value of ΔSBO^{10},I^{10} and the differential table of AES S-box, 2^8 key candidates for each column are left after Step 1.

By considering these details, the 2-round decryption in Step 2-1 of Alg. 2 can be reduced to a 1-round decryption. We notice that both the candidates of K^{10} and corresponding I^{10} can be stored in Step 1, while previous methods only store the candidates of K^{10}. With the stored I^{10}, we do not need to perform the 10^{th} round decryption again in Step 2-1, so that we can reduce a half of calculation time. However, as a penalty, a double size of memory are required to store the results of Step 1.

Divide-and-Conquer attack for Step 2-1 In this attack, we first pre-compute all possible differences of I^9. Then, Step 2-1 of Alg. 2 can be divided into two phases, 2^{30} 1-round decryptions (Step 2-1A) and 2^{26} 1-round decryptions (Step 2-1B).

In Step 0 of Alg. 1, we guess the error propagation path. In the pre-computation phase, for each byte-position which results in the guessed error propagation path, we consider all 2^8 differences and compute corresponding differences of I^9. Store the resultant $2^8 \cdot 4$ or 2^{10} differences in the list G called *valid list*.

In Step 2-1A, we compute a 1-round decryption for I^{10} and $\overline{I^{10}}$ ($=SB^{-1} \circ SR^{-1}(K^{10} \oplus \overline{C})$) to obtain the differences of 2 bytes in I^9. Without losing the generality, we discuss the case where guessed error propagation path is "1" in Fig. 1 and thus 2^{10} ($\Delta I^9_{0,0}$, $\Delta I^9_{0,1}$, $\Delta I^9_{0,2}$, $\Delta I^9_{0,3}$) are stored in G. Moreover, we focus on $\Delta I^9_{0,1}$ and $\Delta I^9_{0,2}$ in Step 2-1A. In general, ΔI^9 is calculated as

$$\begin{aligned} \Delta I^9 = & SB^{-1} \circ SR^{-1} \circ MC^{-1}(K^9 \oplus I^{10}) \\ & \oplus SB^{-1} \circ SR^{-1} \circ MC^{-1}(K^9 \oplus \overline{I^{10}}). \end{aligned} \tag{3}$$

In order to calculate $\Delta I^9_{1,0}$ and $\Delta I^9_{2,0}$, we need $K^9_{2,j}$, $K^9_{3,j}$, $I^{10}_{2,j}$, $I^{10}_{3,j}$, $\overline{I^{10}_{2,j}}$ and $\overline{I^{10}_{3,j}}$ for $j = 0, 1, 2, 3$. From Fig. 2, an attacker can calculate $K^9_{2,j}$ and $K^9_{3,j}$ by Eq. (4).

$$K^9_{2,j} = K^{10}_{2,j} \oplus K^{10}_{1.j}, \quad K^9_{3,j} = K^{10}_{3,j} \oplus K^{10}_{2.j}. \tag{4}$$

Moreover, $I^{10}_{2,j}$, $I^{10}_{3,j}$ can be calculated by Eq. (5).

$$\begin{aligned} I^{10}_{2,j} &= SB^{-1} \circ SR^{-1}(K^{10}_{(3\cdot j+2)\bmod 4,j} \oplus C_{(3\cdot j+2)\bmod 4,j}), \\ I^{10}_{3,j} &= SB^{-1} \circ SR^{-1}(K^{10}_{3-j,j} \oplus C_{3-j,j}). \end{aligned} \tag{5}$$

Similarly, $\overline{I^{10}_{2,j}}$ and $\overline{I^{10}_{3,j}}$ can be calculated with $\overline{C}$. According to Eqs. (4) and (5), 14 bytes of K^{10} except $K^{10}_{0,0}$ and $K^{10}_{0,1}$ are necessary to calculate $\Delta I^9_{1,0}$ and $\Delta I^9_{2,0}$ at Step 2-1A.

For each valid ΔSBO^{10}, two candidates for each byte are considered. However, in this attack, we do not have to consider $K^{10}_{0,1}$ and $K^{10}_{0,2}$ of the key candidates. Hence, the number of candidates of 14 bytes of K^{10} is 2^{30} at Step 1. Consequently, the analysis time of calculating $\Delta I^9_{1,0}$ and $\Delta I^9_{2,0}$ is 2^{30}.

We explain the number of K^{10} candidates after Step 2-1A. When we perform a 1-round decryption with wrong keys, $\Delta I^9_{0,1}$ and $\Delta I^9_{0,2}$ can have all 2^{16} values. Hence, the probability that these match one of the 2^{10} entries in G is $2^{10} \cdot 2^{-16} = 2^{-6}$. Finally, $2^{30} \cdot 2^{-6} = 2^{24}$ candidates of 14 bytes of K^{10} will remain. As a result, we have $2^{24} \cdot 2^2 = 2^{26}$ candidates of 16-bytes of K^{10} after Step 2-1A. Next, we perform Step 2-1B. In this step, an attacker performs Step 2-1 of Alg. 2 for 2^{26} candidates of K^{10} with an improvement shown in Sect. 3.2.

Computational complexity of the method in Sect. 3.2 As can be seen from Table 2, Steps 1, 2-1B and 2-2B of the method proposed in Sect 3.2 cannot be critical operations since all of them are at least two orders of magnitude less than Step 2-1A in computational complexity. Therefore, we focus on Step 2-1A that consists of the following computations. First, we need to pre-compute the differential values for MCO^8, which takes $2^{10} \cdot t\_MC$ where t_MC stands for the operation time of MixColumns. Second, we perform a 1-round decryption with both C and $\overline{C}$ all of the 2^{30} key candidates and check whether the differential is in the pre-computed MCO^8 table, which costs $(2 \cdot t\_AR + t\_KR + t\_CMP\_2^{10}) \cdot 2^{30}$. Here, we perform KR once for a 1-round decryption (AR) of C and $\overline{C}$.

From this observation, we find that the pre-computation time can be ignored and V_3 in Table 2 decides the total operation time in Step 2-1A. As a result, the total time necessary for this method is found to be $2.5 \cdot V_3$ and significant improvements are obtained as shown in Table 2.

3.3 Further improvement of the method in Sect. 3.2 using the linearity of MC

This section improves the attack method in Sect. 3.2. We focus on the property that $K^{10}_{0,2}$ and $K^{10}_{0,3}$ are not related to the last two columns of K^9. Hence, we can independently simulate the impact of these two bytes and update them to the valid list G. This enables us to divide Step 2-1A into two phases as described in Alg. 3.

Algorithm 3. Our improved procedure for Step 2

Step 2-1Aa. For 2^2 candidates of $K^{10}_{0,2}$ and $K^{10}_{0,3}$, update the valid list G.
Step 2-1Ab. For 2^{28} candidates of 12-bytes of K^{10} ($K^{10}_{1,j}, K^{10}_{2,j}, K^{10}_{3,j}$, for j=0 to 3) apply the divide-and-conquer attack explained in Sect. 3.2. 2^{26} candidates of K^{10} will remain after this step.
Step 2-1B. For 2^{26} candidates, apply the improved procedure of Step 2-1 explained in Sect. 3.2. 2^{10} candidates will remain after this step.
Step 2-2. Recover a secret key by the brute force attack for 2^{10} candidates.

In Sect. 3.2, we pre-computed G containing 2^{10} entries of $(\Delta I^9_{0,1}, \Delta I^9_{0,2})$. In this attack, we further perform the pre-computation by considering all 2^{16} values of $(I^9_{0,1}, I^9_{0,2})$ for each stored $(\Delta I^9_{0,1}, \Delta I^9_{0,2})$. We then obtain 2^{26} items of $(\Delta MCI^9_{2,2}, \Delta MCI^9_{3,1}, MCI^9_{2,2}, MCI^9_{3,1})$ and store them in G. Notice that for all 2^{16} $(\Delta MCI^9_{2,2}, \Delta MCI^9_{3,1})$, we will obtain 2^{10} items of $(MCI^9_{2,2}, MCI^9_{3,1})$. Also note that we need a memory with 2^{26} entries in the pre-computation phase. However, candidates of $(\Delta MCI^9_{2,2}, \Delta MCI^9_{3,1})$ are reduced to 2^8 in Step 1 of Alg. 1. Hence, in on-line, only 2^{18} entries in G are needed.

In this attack, the validity of the candidates of K^{10} is checked by computing $(\Delta MCI^9_{2,2}, \Delta MCI^9_{3,1}, MCI^9_{2,2}, MCI^9_{3,1})$ using Eq. (6) for $i = 2, 3$ and matching them with entries in G.

$$MC^{-1}(K^9_{i,0} \oplus I^{10}_{i,0}, K^9_{i,1} \oplus I^{10}_{i,1}, K^9_{i,2} \oplus I^{10}_{i,2}, K^9_{i,3} \oplus I^{10}_{i,3})^T. \tag{6}$$

By considering the details and linearity of MC, the match of $MCI^9_{2,2}$ using Eq. (6) can be written as follows (We denote the N-th byte of a column A by $A[N]$).

$$\begin{aligned} MCI^9_{2,2} \stackrel{?}{=} & MC^{-1}(K^9_{2,0}, K^9_{2,1}, K^9_{2,2}, K^9_{2,3})^T[2] \\ & \oplus (13 \cdot I^{10}_{2,0}) \oplus (9 \cdot I^{10}_{2,1}) \oplus (14 \cdot I^{10}_{2,2}) \oplus (11 \cdot I^{10}_{2,3}). \end{aligned}$$

Because $I^{10}_{2,2}$ is not related to the value of $K^9_{2,0}, K^9_{2,1}, K^9_{2,2}, K^9_{2,3}$, we consider the following form (do the same for $MCI^9_{3,1}$).

$$\begin{aligned} MCI^9_{2,2} \oplus 14 \cdot I^{10}_{2,2} \stackrel{?}{=} & MC^{-1}(K^9_{2,0}, K^9_{2,1}, K^9_{2,2}, K^9_{2,3})^T[2] \\ & \oplus 13 \cdot I^{10}_{2,0} \oplus 9 \cdot I^{10}_{2,1} \oplus 11 \cdot I^{10}_{2,3}, \\ MCI^{10}_{3,1} \oplus 14 \cdot I^{10}_{3,3} \stackrel{?}{=} & MC^{-1}(K^9_{3,0}, K^9_{3,1}, K^9_{3,2}, K^9_{3,3})^T[3] \\ & \oplus 11 \cdot I^{10}_{3,0} \oplus 13 \cdot I^{10}_{3,1} \oplus 9 \cdot I^{10}_{3,2}. \end{aligned} \tag{7}$$

With Eq. (7), we can compute the impact of $I^{10}_{2,2}$ and $I^{10}_{3,3}$ independently of other 12-byte values. The attack procedure is as follows.

For all 2^8 candidates of $(\Delta MCI^9_{2,2}, \Delta MCI^9_{3,1})$, do as follows:
- For 2 candidates of $I^{10}_{2,2}$, compute $14 \cdot I^{10}_{2,2}$. Denote these values by u_0 and u_1.
- For 2 candidates of $I^{10}_{3,3}$, compute $14 \cdot I^{10}_{3,3}$. Denote these values by v_0 and v_1.
- For all 2^{10} entries in the list G for the same $(\Delta MCI^9_{2,2}, \Delta MCI^9_{3,1})$, compute $(MCI^9_{2,2} \oplus u_i, MCI^9_{3,1} \oplus v_j)$ for $0 \le i, j \le 1$. Update G with these 2^{12} entries.

Finally, we can compute the right side of Eq. (7) with only considering 12 bytes of K^{10}. The attack complexity is 2^{20} for updating G and 2^{28} for checking the match for all 12-byte candidates of K^{10}. Hence, the total complexity is $2^{20}+2^{28} \approx 2^{28}$. After the match, 2^{26} key candidates will be left and the remaining attack procedure is the same as Section 3.2. Note that a memory with 2^{20} entries is used at on-line phase.

Computational complexity of the method in Sect. 3.3 Only the difference between the methods in Sects. 3.2 and 3.3 is in Step 2-1A. As can be seen from Table 2, the computational complexities of Steps 2-1A and 2-1B in this method are getting close. Therefore, we need to consider both Steps 2-1A and Steps 2-1B.

As for Step 2-1A, we pre-compute the list G containing 2^{20} entries. The pre-computed table is calculated based on 2^{16} values of $(I^9_{0,1}, I^9_{0,2})$, which results in the computational complexity of much less than the complexity of the main calculation of Step 2-1A. The main calculation performs two AR and one KR to derive the corresponding value to be checked with ones in list G for all 2^{28} candidates. Consequently, the computational step in Step 2-1A becomes $V_4 = (2 \cdot t\_AR + t\_KR + t\_CMP\_2^{20}) \cdot 2^{28}$. Step 2-1B is the same as the method in Sect. 3.2, where we perform further reduction of the remaining 2^{26} key candidates by utilizing the AES key schedule. More precisely, this step computes the differential of MCI^8 for C and $\overline{C}$ and checks whether if the differential has only 1-byte error, which results in the computational cost of $V_5 = (t\_CMP + 2 \cdot t\_AR + t\_KR) \cdot 2^{26}$. As a result, the total time of this method is $2.5 \cdot (V_4 + V_5)$ and further improvements are obtained as shown in Table 2.

4 Conclusion and Future Work

Based on the detailed analyses of the previous DFA methods and the consideration of practical implementations, this paper proposed improved analysis methods for Differential Fault Analysis (DFA) against AES. The expectation of our fastest algorithm only costs about one sixteenth of that of the fastest previous analysis methods. The improvements come from three new techniques as the combination of previous DFA methods, the divide-and-conquer attack by considering the AES key-schedule, and taking the linearity of MixColumns operation into consideration. This paper applied our proposed techniques to the case where 1-byte error is injected at the beginning of 8^{th} round. In the future, we will apply these techniques to the cases where multiple-byte errors are injected into AES.

References

1. E. Biham and A. Shamir, "Differential Fault Analysis of Secret Key Cryptosystems," in *Advances in Cryptology – CRYPTO '97*, vol. 1294 of *LNCS*, pp. 513–525, Springer, 1997.
2. J. Blömer and J.-P. Seifert, "Fault Based Cryptanalysis of the Advanced Encryption Standard (AES)," in *Financial Cryptography FC 2003*, vol. 2742 of *LNCS*, pp. 162–181, Springer Berlin, 2003.
3. P. Dusart, G. Letourneux, and O. Vivolo, "Differential Fault Analysis on A.E.S," in *Applied Cryptography and Network Security, First International Conference, ACNS 2003. Kunming, China, October 16-19, 2003*, LNCS, pp. 293–306, Springer, 2003.
4. C. Giraud, "DFA on AES," in *Advanced Encryption Standard - AES, 4th International Conference, AES 2004, Bonn, Germany, May 10-12, 2004, Revised Selected and Invited Papers*, vol. 3373 of *LNCS*, pp. 27–41, Springer, 2005.
5. G. Piret and J.-J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and Khazad," in *Cryptographic Hardware and Embedded Systems – CHES 2003*, vol. 2779 of *LNCS*, pp. 77–88, Springer, 2003.
6. D. Mukhopadhyay, "An Improved Fault Based Attack of the Advanced Encryption Standard," in *AFRICACRYPT 2009*, vol. 5580 of *LNCS*, pp. 421–434, Springer, 2009.
7. D. Saha, D. Mukhopadhyay, and D. RoyChowdhury, "A Diagonal Fault Attack on the Advanced Encryption Standard," 2009. Cryptology ePrint Archive, Report2009/581.
8. M. Tunstall and D. Mukhopadhyay, "Differential Fault Analysis of the Advanced Encryption Standard using a Single Fault," 2009. Cryptology ePrint Archive, Report2009/575.
9. National Institute of Standards and Technology, "Advanced Encryption Standard." NIST FIPS PUB 197, 2001.