New Differential Fault Analysis on AES Key Schedule: Two Faults Are Enough

Chong Hee Kim* and Jean-Jacques Quisquater

UCL Crypto Group, Université Catholique de Louvain, Belgium
Place du Levant, 3, Louvain-la-Neuve, 1348, Belgium
{chong-hee.kim,Jean-Jacques.Quisquater}@uclouvain.be

Abstract. In this paper we show a new differential fault analysis (DFA) on the AES-128 key scheduling process. We can obtain 96 bits of the key with 2 pairs of correct and faulty ciphertexts enabling an easy exhaustive key search of 2^{32} keys. Furthermore we can retrieve the entire 128 bits with 4 pairs. To the authors' best knowledge, it is the smallest number of pairs to find the entire AES-128 key with a fault attack on the key scheduling process. Up to now 7 pairs by Takahashi et al. were the best. By corrupting state, not the key schedule, Piret and Quisquater showed 2 pairs are enough to break AES-128 in 2003. The advantage of DFA on the key schedule is that it can defeat some fault-protected AES implementations where the round keys are not rescheduled prior to the check. We implemented our algorithm on a 3.2 GHz Pentium 4 PC. With 4 pairs of correct and faulty ciphertexts, we could find 128 bits less than 2.3 seconds.

Index terms: Fault attack, Differential Fault Analysis, AES, DFA, AES key schedule.

1 Introduction

Boneh et al. introduced the *fault attack* on the implementation of RSA-CRT (Chinese Remainder Theorem) with the errors induced by the fault injection in September 1996 [5]. After that, many papers have been published on this subject. In October 1996, Biham and Shamir published a fault attack on secret key cryptosystems entitled *Differential Fault Analysis* (DFA) [2]. On the 2^{nd} October 2000, the AES became the successor of the DES and since then, it has been used more and more in many applications. Several authors mounted DFA on AES [3,7,9,11]. They assumed that the intermediate states were corrupted by the fault injection and tried to find out the key. Among them, the attack by Piret and Quisquater is the most efficient [11]. Their attack only needs two pairs of correct and faulty ciphertexts to retrieve 128 bits of AES-128.

A different form of DFA, targeting the AES key schedule, was introduced by Giraud in [8] and improved by Chen and Yen [6]. However, Giraud's attack does

* Supported by Walloon Region, Belgium / E.USER project.

G. Grimaud and F.-X. Standaert (Eds.): CARDIS 2008, LNCS 5189, pp. 48–60, 2008.

not target only the AES key schedule. He used the faults in the intermediate state as well. Recently Peacham and Thomas improved DFA on AES key scheduling [10]. They assumed that random faults are injected during the execution of the AES key scheduling process and that the resulting faults propagate to all keys after the injection. They reduced the number of correct and faulty ciphertexts to 12 pairs to recover 128-bit key. Takahashi et al. generalized Peacham and Thomas's attack and reduced the required number of pairs more [13]. They succeeded in recovering 128-bit key with 7 pairs.

In this paper, we propose a new DFA on AES key scheduling process. Our attack takes advantage of faults occurring in the 9^{th} round of the AES key scheduling process. Thus the fault model and the hypothesis on the fault location are exactly the same as in Peacham and Thomas' and Takahashi et al.'s. However the way we exploit faults is different from theirs. We retrieved the entire 128-bit key of AES-128 with 4 pairs of correct and faulty ciphertexts. Two pairs are enough to recover 96 bits of the key enabling an easy exhaustive key search of the remaining 2^{32} keys. We implemented our algorithm on a 3.2 GHz Pentium 4 PC. With 4 pairs of correct and faulty ciphertexts we found 128 bits in less than 2.3 seconds.

The rest of this paper is organized as follows. In Section 2, we briefly describe AES. The review on the previous DFA on AES Key schedule is presented in Section 3. Our analysis methodology is presented in Section 4. Section 5 compares our attack with previous attacks, with the conclusion given in Section 6.

2 AES

AES [1] can encrypt and decrypt 128 bits of block with 128, 192, or 256 bits of key. In this paper we will only deal with the 128-bit key variant, AES-128, as it is the most widely used. Our attack can be extended trivially to other variants. The intermediate computation result of AES-128, called *State* is usually represented by a 4×4 matrix, each cell of which is a byte as shown in Fig. 1. Where, $S^i_{j,k}$ denotes $(j+1)^{th}$ row and $(k+1)^{th}$ column byte of i^{th} State, $j, k \in \{0, \ldots 3\}$.

In the rest of the paper, we will use the following additional notations:

- $K^i_{j,k}$ denotes $(j+1)^{th}$ row and $(k+1)^{th}$ column byte of i^{th} AES round key, $i \in \{0, \ldots, 10\}$ and $j, k \in \{0, \ldots 3\}$,
- S^0 denotes the State after 9^{th} *AddRoundKey*,
- S^1 denotes the State after 10^{th} *SubBytes*,
- S^2 denotes the State after 10^{th} *ShiftRows*,
- S^3 denotes the State after 10^{th} *AddRoundKey*,
- S^i_j denotes 32-bit $(j+1)^{th}$ row of S^i, $j \in \{0, \ldots 3\}$,
- (C, C^*), (D, D^*) denote the correct and faulty ciphertext pairs.

AES-128 has 10 rounds. Each round function is composed of 4 transformations except the last round: *SubBytes*, *ShiftRows*, *MixColumns*, and *AddRoundKey*.

$S^i_{0,0}$	$S^i_{0,1}$	$S^i_{0,2}$	$S^i_{0,3}$
$S^i_{1,0}$	$S^i_{1,1}$	$S^i_{1,2}$	$S^i_{1,3}$
$S^i_{2,0}$	$S^i_{2,1}$	$S^i_{2,2}$	$S^i_{2,3}$
$S^i_{3,0}$	$S^i_{3,1}$	$S^i_{3,2}$	$S^i_{3,3}$

Fig. 1. The State during AES encryption

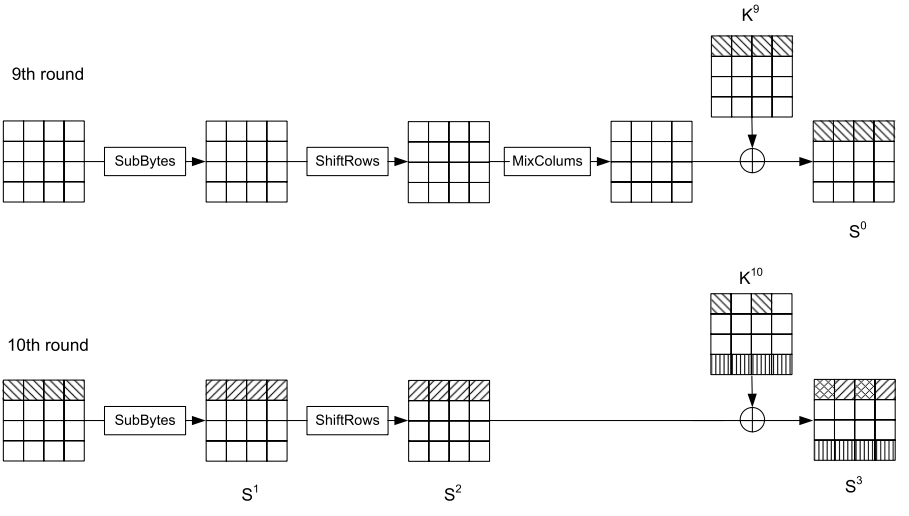

Fig. 2. Last two rounds of AES encryption

The last round is lacking *MixColumns*. Our attack focuses on the last two rounds. They are depicted in Fig. 2.

SubBytes. It is made up of the application of 16 identical 8×8 S-boxes. This is a non-linear byte substitution. We denote the function of SubBytes **SB**. That is, $\mathbf{SB}(S^i) = \text{SubBytes}(S^i)$. For the simplicity, we define that **SB** also can take a byte and two bytes as an input as follows:

$\mathbf{SB}(S^i_{j,k}) = \text{Sbox}(S^i_{j,k})$,
$\mathbf{SB}(S^i_{j,k}, S^i_{j,l}) = \text{Sbox}(S^i_{j,k}), \text{Sbox}(S^i_{j,l})$.

We denote *Inverse SubBytes* $\mathbf{SB^{-1}}$. We define that $\mathbf{SB^{-1}}$ also can take bytes as an input.

ShiftRows. Each row of the *State* is cyclically shifted over different offsets. Row 0 is not shifted, row 1 is shifted by 1 byte, row 2 is shifted by 2 bytes, and row 3 by 3 bytes. We denote *ShiftRows* and its inverse, *InverseShiftRows*, **SR** and $\mathbf{SR^{-1}}$ respectively. We also define that they can take bytes as an input.

MixColumns. This is a linear transformation to each column of the *State*. Each column is considered as polynomial over $\mathbb{F}_{2^8}$ and multiplied modulo x^4+1 with a fixed polynomial $a(x) = 03 * x^3 + 01 * x^2 + 01 * x + 02$.

AddRoundKey. It is a bitwise XOR with a round key.

We briefly describe the last two rounds, 9^{th} and 10^{th} rounds, of the key scheduling process as shown in Fig. 3. The input 128-bit key is divided into four 32-bit columns. The first 32-bit column propagates to the next column in the same round, which generates the second column. The second and third columns do the same. The fourth column propagates to the next round through the function RotWord, which performs a cyclic permutation and SubWord, which applies S-box. Each column generated in the key process is grouped to yield the 128-bit round key.

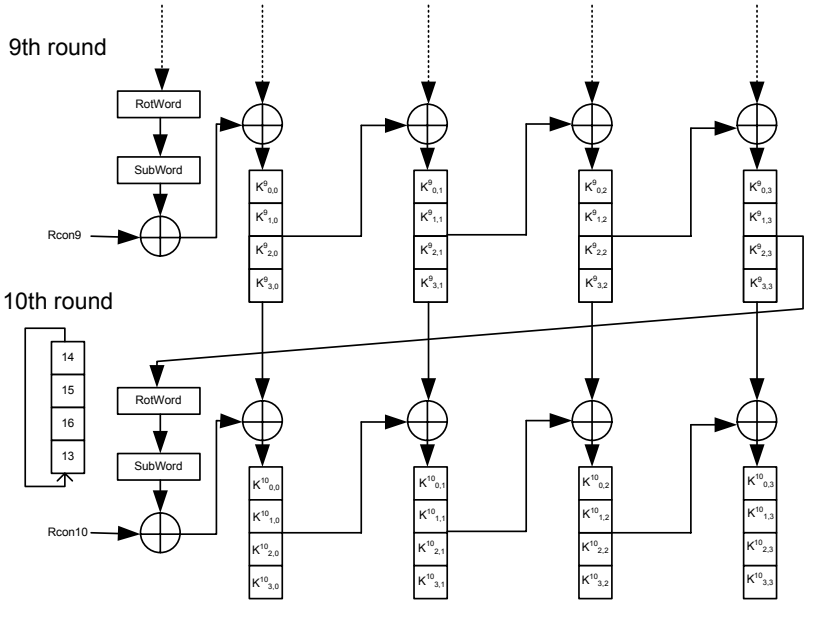

Fig. 3. 9^{th} and 10^{th} AES Key scheduling process

3 Previous Works about DFA on AES Key Schedule

The first DFA on AES key schedule was done by Giraud, but still needs to attack the intermediate state [8]. He presented two fault attacks on the AES. Both

require the ability to obtain several faulty ciphertexts originating from the *same* plaintext (contrary to our attack). The first one assumes it is possible to induce a fault on only one bit of an intermediate state. Under this condition, 50 faulty ciphertexts are necessary to retrieve the full key. The second attack exploits faults on bytes. It requires the ability of inducing faults at several chosen places both on key scheduling process and intermediate state. Therefore, the second is the first attempt of attack on key scheduling process, but it is not complete. Because it still needs to attack on the intermediate state. It could retrieve the key with 250 faulty ciphertexts. If he extends his hypothesis by supposing that the attacker can choose the byte affected by the fault, the first attack requires 35 faulty ciphertexts and the second requires 31 faulty ciphertexts.

In 2003, Chen and Yen improved Giraud's attack [6]. They could retrieve the key with fewer faulty ciphertexts and with less computational complexity. Giraud's second attack is composed of three steps, an attack on 9^{th} round key, an attack on 8^{th} round key, and an attack on 8^{th} round intermediate state. The first two steps of Chen and Yen's method are similar to Giraud's. But the third step focuses on the *Inverse SubBytes* and requires less samples.

Unlike the previous two attacks, Peacham and Thomas assumes that random faults are injected during the execution of the AES key scheduling process and the resulting faults propagate to all keys after the injection [10]. They showed that 12 pairs of correct and faulty ciphertexts are enough to retrieve the whole key without brute-force search. They assumed that all bytes of a 32-bit column of the 9^{th} round key are corrupted during the execution of the key scheduling process. Their attack consists of four steps. They use the fact that the intermediate state calculated by the correct ciphertext just before the *AddRoundKey* of the 9^{th} round is equal to the intermediate state calculated by the faulty ciphertext just before the *AddRoundKey* of the 9^{th} round.

In 2007, Takahashi et al. generalized Peacham and Thomas's attack and reduced the required number of pairs [13]. They could retrieve the whole key with 7 pairs and 80 bits of the key with 2 pairs. Recently they improved their attack a little bit by assuming that faults are injected into 32 bits of one column [12]. They found 88 bits with 2 pairs but still they need 7 pairs to find 128 bits.

4 Our DFA on AES Key Schedule

In this section we describe our attack. After presenting our fault model, we describe our *basic attack* that retrieves 32 bits of the key with 2 pairs of correct and faulty ciphertexts giving a one byte random error on 9^{th} round key scheduling process. Then we improve our attack by giving a random fault on three bytes of a 32-bit column of the 9^{th} round key scheduling process. We can retrieve 96 bits of the key with 2 pairs, and all 128 bits with 4 pairs.

4.1 Fault Model

We assume that a random fault is induced in the 9^{th} round of the AES key scheduling process and some bytes of the first column of the 9^{th} round key are

corrupted. In addition, we assume that the attacker can obtain pairs of correct and faulty outputs from the same input. However we do not need the several faulty outputs with the same plaintext.

The fault model and hypothesis on the fault location are exactly the same as in Peacham and Thomas' and Takahashi et al.'s. However the way we exploit faults is different from them. We exploit the intermediate state *after* the *AddRoundKey* of the 9^{th} round contrary to previous attacks. They exploit the intermediate state *before* the *AddRoundKey* of the 9^{th} round. Secondly they try to find the 9^{th} round key but we find the 10^{th} round key. Finally we try to remove impossible candidates for 10^{th} round key, but they try to find directly the correct 9^{th} round key.

It is quite interesting to refer DFA on AES state. Piret and Quisquater's DFA on AES state [11] requires the minimum number of the ciphertexts even though they use the same fault model and the hypothesis on the fault location of Dusart et al.'s attack [7]. The way Dusart et al. exploit faults is quite similar to Peacham and Thomas' and Takahashi et al.'s. Dusart et al. write and solve a system of equations of which the unknown value is the one of the fault. Our way of exploiting faults follows the way of Piret and Quisquater's.

4.2 Basic Attack

We assume that one byte of the first column of the 9^{th} round key is corrupted. For simplicity, we assume $K^9_{0,0}$ is corrupted into $\widetilde{K}^9_{0,0}$. The attack can be applied when another byte is corrupted. We denote the difference between them as a, i.e., $a = K^9_{0,0} \bigoplus \widetilde{K}^9_{0,0}$. This error propagates to some bytes of the round keys as shown in Fig. 4. The four bytes of 9^{th} round key, $(K^9_{0,0}, K^9_{0,1}, K^9_{0,2}, K^9_{0,3})$, and six bytes of 10^{th} round key, $(K^{10}_{0,0}, K^{10}_{0,2}, K^{10}_{3,0}, K^{10}_{3,1}, K^{10}_{3,2}, K^{10}_{3,3})$, are corrupted. This also results in the corruption on the intermediate states as shown in Fig. 2.

We exploit the intermediate state *after* the *AddRoundKey* of the 9^{th} round, i.e., S^0. We denote the i^{th} faulty state $\widetilde{S}^i$. Only the first row of S^0 receives the effect of the faults. By doing XOR between S^0_0 and $\widetilde{S}^0_0$, we have (we remind that S^i_j is denoted as a 32-bit $(j+1)^{th}$ row of S^i):

$$\begin{aligned} S^0_0 \oplus \widetilde{S}^0_0 &= \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C_0 \oplus K^{10}_0)] \oplus \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C^*_0 \oplus \widetilde{K}^{10}_0)] \\ &= (a, a, a, a), \end{aligned} \tag{1}$$

where, C_0 is the 32-bit first row of the correct ciphertext, C^*_0 is the 32-bit first row of the faulty ciphertext, K^{10}_0 is the 32-bit first row of 10^{th} correct round key, and $\widetilde{K}^{10}_0$ is the 32-bit first row of 10^{th} faulty round key.

An error on a byte makes 255 possible differences. That is, $a \in \{1, 2, \ldots, 255\}$. Therefore the number of the possible differences of S^0_0 and $\widetilde{S}^0_0$ is 255. In equation (1), we know C_0 and C^*_0. Therefore we can eliminate the wrong candidates for K^{10}_0 that do not meet the condition of equation (1).

We compute how many wrong candidates for the round key K^{10}_0 can be removed by a single pair (C, C^*) with the equation (1). We define the difference of the first 32-bit row of C and C^* as Δ, i.e., $\Delta = C_0 \oplus C^*_0$. The number of

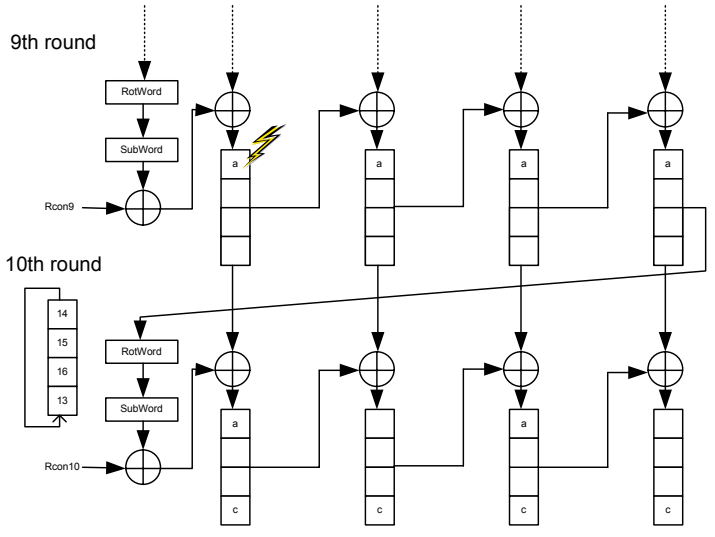

Fig. 4. 9^{th} and 10^{th} AES Key scheduling process with faults

possible value for Δ is 255^4. Among them 255 differences satisfy the equation (1). Thus the fraction of the candidates for the round key K_0^{10} surviving the test with equation (1) is $255/255^4$. Therefore we conclude that the number of remaining wrong candidates for K_0^{10} after N pairs have been treated is about $256^4(255^{-3})^N$. With one pair, about 259 candidates remain. If two pairs are exploited, we are in principle left with the right candidate only.

In the above paragraph we assumed that we needed to guess K_0^{10} only and we knew other parameters. However, to solve the equation (1) we need to guess both K_0^{10} and $\widetilde{K}_0^{10}$. Since we have 2^{64} candidates for $(K_0^{10}, \widetilde{K}_0^{10})$, it is not practical to guess K_0^{10} and $\widetilde{K}_0^{10}$ together. However, K_0^{10} and $\widetilde{K}_0^{10}$ satisfy the following condition:

$$K_0^{10} \oplus \widetilde{K}_0^{10} = (a, 0, a, 0).$$

That is, $K_{0,1}^{10} = \widetilde{K}_{0,1}^{10}$ and $K_{0,3}^{10} = \widetilde{K}_{0,3}^{10}$. Therefore we can start the attack with these two bytes that results in 2^{16} candidates instead of 2^{64}. The number of remaining candidates for $(K_{0,1}^{10}, K_{0,3}^{10})$ after N pairs of the correct and wrong ciphertexts is now $256^2(255^{-1})^N$. With two pairs, we are left with almost one candidate for $(K_{0,1}^{10}, K_{0,3}^{10})$[1].

This consideration leads to the following sketch of our *basic attack*. We need two pairs of the correct and faulty ciphertexts (C, C^*) and (D, D^*). We do not need to have the same faulty value in $K_{0,0}^9$ for these two pairs. We define the

[1] Because $256^2(255^{-1})^2 = 1.0079$, we have more than one candidate left for $(K_{0,1}^{10}, K_{0,3}^{10})$ sometimes. However, after Step 2 we have only one candidate left since $256^3(255^{-2})^2 = 0.004$ for $(K_{0,1}^{10}, K_{0,2}^{10}, K_{0,3}^{10})$.

error during the computation of C^* as a_1, i.e., $a_1 = K^9_{0,0} \oplus \widetilde{K}^9_{0,0}$ and the error during the computation of D^* as a_2, i.e., $a_2 = K^9_{0,0} \oplus \widetilde{K}^9_{0,0}$ for (D, D^*).

We first find the candidates for $(K^{10}_{0,1}, K^{10}_{0,3})$ with two pairs of the correct and faulty ciphertexts. Normally after this step, we have 1 or 2 candidates. Then we find the candidates for $(K^{10}_{0,1}, K^{10}_{0,2}, K^{10}_{0,3})$. Finally we find the candidates for $(K^{10}_{0,0}, K^{10}_{0,1}, K^{10}_{0,2}, K^{10}_{0,3})$.

Step 1. We compute the candidate for $(K^{10}_{0,1}, K^{10}_{0,3}, a_1, a_2)$. The inputs for this step are two pairs of the correct and faulty ciphertexts (C, C^*) and (D, D^*).

Algorithm 1

1. Set up a list $\mathcal{L}$ containing all 2^{16} candidates for $(K^{10}_{0,1}, K^{10}_{0,3})$.
2. Choose a candidate from $\mathcal{L}$ and compute (α_1, α_2) and (β_1, β_2) as follows: $(\alpha_1, \alpha_2) = \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C_{0,1} \oplus K^{10}_{0,1}, C_{0,3} \oplus K^{10}_{0,3})] \oplus \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C^*_{0,1} \oplus K^{10}_{0,1}, C^*_{0,3} \oplus K^{10}_{0,3})]$ and $(\beta_1, \beta_2) = \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(D_{0,1} \oplus K^{10}_{0,1}, D_{0,3} \oplus K^{10}_{0,3})] \oplus \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(D^*_{0,1} \oplus K^{10}_{0,1}, D^*_{0,3} \oplus K^{10}_{0,3})]$.
3. Add the candidate and (α_1, β_1) to a new list $\mathcal{M}$ if $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$.
4. Repeat Step 2 and Step 3 for all candidates from $\mathcal{L}$.

Finally $\mathcal{M}$ has the candidates for $(K^{10}_{0,1}, K^{10}_{0,3}, a_1, a_2)$.

Step 2. We compute the candidate for $(K^{10}_{0,1}, K^{10}_{0,2}, K^{10}_{0,3}, a_1, a_2)$. The inputs for this step are (C, C^*), (D, D^*), and the list $\mathcal{M}$ from Step 1. We note that $\widetilde{K}^{10}_{0,2}$ can be computed as $\widetilde{K}^{10}_{0,2} = K^{10}_{0,2} \oplus a$.

Algorithm 2

1. Set up a list $\mathcal{L}$ containing all 2^8 candidates for $K^{10}_{0,2}$.
2. Choose a candidate from $\mathcal{L}$.
3. Choose a candidate from $\mathcal{M}$ and compute (α_1, α_2) and (β_1, β_2) as follows: $(\alpha_1, \alpha_2) = \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C_{0,2} \oplus K^{10}_{0,2}, C_{0,3} \oplus K^{10}_{0,3})] \oplus \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C^*_{0,2} \oplus K^{10}_{0,2} \oplus a_1, C^*_{0,3} \oplus K^{10}_{0,3})]$ and $(\beta_1, \beta_2) = \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(D_{0,2} \oplus K^{10}_{0,2}, D_{0,3} \oplus K^{10}_{0,3})] \oplus \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(D^*_{0,2} \oplus K^{10}_{0,2} \oplus a_2, D^*_{0,3} \oplus K^{10}_{0,3})]$.
4. Add $(K^{10}_{0,1}, K^{10}_{0,2}, K^{10}_{0,3}, a_1, a_2)$ to a new list $\mathcal{N}$ if $\alpha_1 = \alpha_2 = a_1$ and $\beta_1 = \beta_2 = a_2$.
5. Repeat Step 3 and Step 4 for all candidates from $\mathcal{M}$.
6. Repeat from Step 2 to Step 5 for all candidates from $\mathcal{L}$.

Finally $\mathcal{N}$ has the candidates for $(K^{10}_{0,1}, K^{10}_{0,2}, K^{10}_{0,3}, a_1, a_2)$.

Step 3. We compute the candidate for $(K^{10}_{0,0}, K^{10}_{0,1}, K^{10}_{0,2}, K^{10}_{0,3}, a_1, a_2)$. The inputs for this step are (C, C^*), (D, D^*), and the list $\mathcal{N}$ from Step 2. We note that $\widetilde{K}^{10}_{0,0}$ can be computed as $\widetilde{K}^{10}_{0,0} = K^{10}_{0,0} \oplus a$.

Algorithm 3

1. Set up a list $\mathcal{L}$ containing all 2^8 candidates for $K^{10}_{0,0}$.
2. Choose a candidate from $\mathcal{L}$.

3. Choose a candidate from $\mathcal{N}$ and compute (α_1, α_2) and (β_1, β_2) as follows: $(\alpha_1, \alpha_2) = \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C_{0,0} \oplus K^{10}_{0,0}, C_{0,3} \oplus K^{10}_{0,3})] \oplus \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C^*_{0,0} \oplus K^{10}_{0,0} \oplus a_1, C^*_{0,3} \oplus K^{10}_{0,3})]$ and $(\beta_1, \beta_2) = \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(D_{0,0} \oplus K^{10}_{0,0}, D_{0,3} \oplus K^{10}_{0,3})] \oplus \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(D^*_{0,0} \oplus K^{10}_{0,0} \oplus a_2, D^*_{0,3} \oplus K^{10}_{0,3})]$.
4. Output $(K^{10}_{0,0}, K^{10}_{0,1}, K^{10}_{0,2}, K^{10}_{0,3}, a_1, a_2)$ and stop the algorithm if $\alpha_1 = \alpha_2 = a_1$ and $\beta_1 = \beta_2 = a_2$.
5. Repeat Step 3 and Step 4 for all candidates from $\mathcal{N}$.
6. Repeat from Step 2 to Step 5 for all candidates from $\mathcal{L}$.

Finally we have the one correct key for $(K^{10}_{0,0}, K^{10}_{0,1}, K^{10}_{0,2}, K^{10}_{0,3})$ and faulty values (a_1, a_2).

We implemented our attack on a 3.2 GHz Pentium 4 PC and found K^{10}_0 with about 0.5 second. If we give faults in $K^9_{1,0}$ instead of $K^9_{0,0}$, then similarly we can find K^{10}_1. Therefore with 8 pairs, we can find the entire 128 bits of 10^{th} round key. We can easily compute the initial key with 10^{th} round key, see [7].

4.3 Improved Attack

Now let us consider the errors on more than one byte on the first column of 9^{th} round key. We suppose that the first two bytes, $K^9_{0,0}$ and $K^9_{1,0}$, are corrupted by fault injection. Let us denote that $a = K^9_{0,0} \oplus \widetilde{K}^9_{0,0}$ and $b = K^9_{1,0} \oplus \widetilde{K}^9_{1,0}$. According to AES key scheduling process, these differences a and b make another difference c and d in 10^{th} round key respectively as shown in Fig. 5.

Two-byte error makes two rows of S^0 to be corrupted. We define the difference in the second row, S^0_1, as (b, b, b, b). The corresponding difference in the second row of the 10^{th} round key, K^{10}_1, is $(b, 0, b, 0)$. Therefore we can find K^{10}_1 and b with the *basic attack* in Sec. 4.2 with the following condition:

$$\mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C_1 \oplus K^{10}_1)] \oplus \mathbf{SB^{-1}}[\mathbf{SR^{-1}}(C^*_1 \oplus \widetilde{K}^{10}_1)] = (b, b, b, b). \qquad (2)$$

Then we can compute d from the value of b and K^{10}_1 (this comes from the structure of the AES keys scheduling process) as follows:

$$\begin{aligned} K^9_{1,3} &= K^{10}_{1,2} \oplus K^{10}_{1,3}, \\ d &= \mathrm{SBox}(K^9_{1,3}) \oplus \mathrm{SBox}(K^9_{1,3} \oplus b). \end{aligned}$$

Because we know the value of d, we only do not know the value of a in the first row of difference of K^{10} as shown in (c) of Fig. 5. Therefore we can apply the *basic attack* to the first row and find K^{10}_0.

We summarize how to find 64 bits of K^{10} as follows:

Algorithm 4

1. Compute K^{10}_1 and (b_1, b_2) using *basic attack*.
2. Compute d_1, d_2 with $(K^{10}_{1,2}, K^{10}_{1,3})$ and b_1, b_2.
3. Compute K^{10}_0 and (a_1, a_2) using *basic attack*.

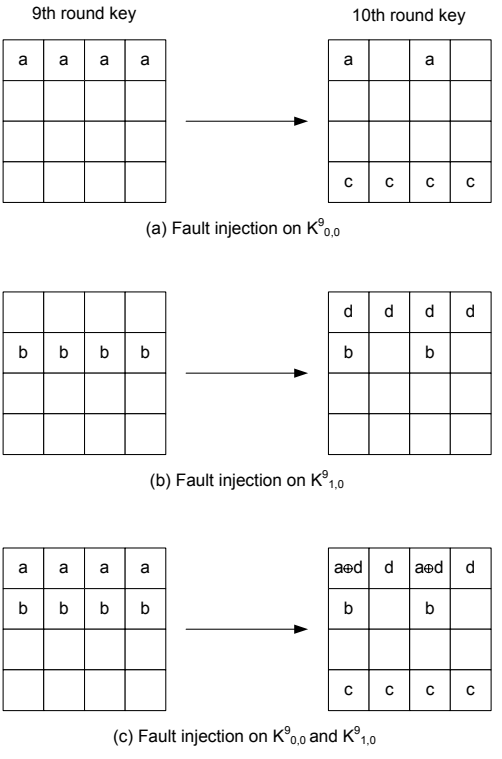

Fig. 5. Differences between correct and wrong round keys according to fault injection on the 9^{th} round key

We can further improve the attack in case three bytes of the first column of 9^{th} round key are corrupted. As shown in Fig. 6, let us denote $e = K^9_{2,0} \oplus \widetilde{K}^9_{2,0}$. We first start with the third row. With the basic attack, we compute K^{10}_2 and e. Then we compute f with the property of AES key scheduling process. We can compute K^{10}_1 and K^{10}_0 with Algorithm 4. Therefore we can compute 96 bits of AES-128 key with two pairs of correct and faulty ciphertexts. We can compute

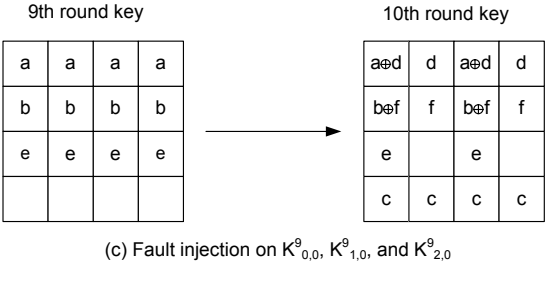

Fig. 6. Differences between correct and wrong round keys in case of the fault injection on the three bytes of the 9^{th} round key

the last 32 bits with an exhaustive search or with the basic attack of one byte fault on $K^9_{3,0}$ and another two pairs of correct and faulty ciphertexts.

We again implemented our improved attack on the same PC. To find 96 bits of the key with 2 pairs, average 1.8 seconds are required. To compute 128 bits with 4 pairs, it requires 2.3 seconds in average.

5 Comparison with Previous Attacks

We compared our attack with previous attacks in terms of the relation between the retrieved bits of key and the required number of pairs as shown in Fig. 7.

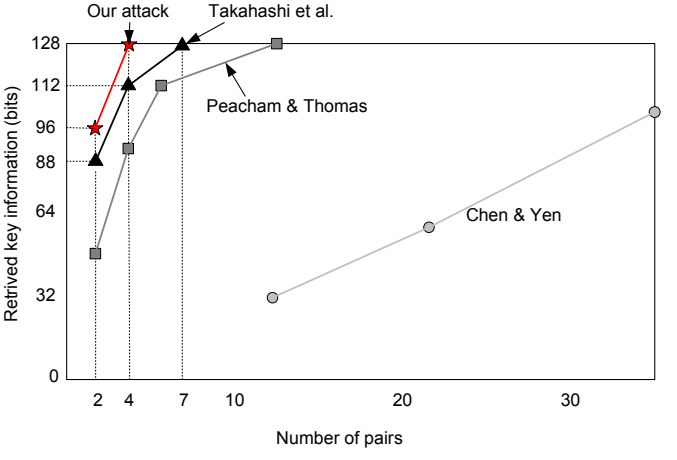

Fig. 7. Comparison in terms of required number of pairs

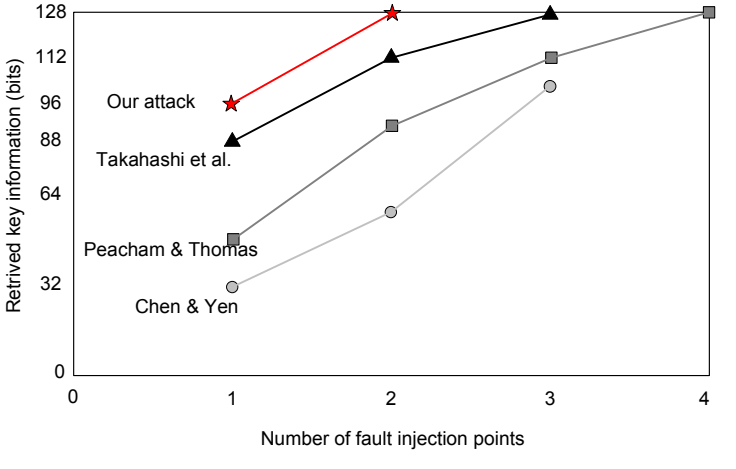

Fig. 8. Comparison in terms of required number of fault injection points

We also compared in terms of the number of fault injection points in Fig. 8. In both cases, we can see our proposed method is the best.

If we have only two pairs of the correct and wrong ciphertexts, we can compute 96 bits with our attack and need to do an exhaustive search for the other 32 bits. We estimated the time for the exhaustive search based on the simulation result of [13]. On a normal PC, the estimated calculation time of the 32-bit exhaustive search is about 8 minutes. If we use Takahashi et al.'s attack, we need to do an 40-bit exhaustive search, which requires about 3 days.

6 Conclusions

We proposed a new differential fault analysis on AES key schedule. Only two pairs of correct and faulty ciphertexts are enough to find the whole key of AES-128 with DFA on AES state by Piret and Quisquater. In the area of DFA on AES key schedule, still we needed many pairs. However our proposed method reduced the gap between DFA on state and DFA on key schedule. Ours requires two pairs for retrieving 96 bits of the key enabling an easy exhaustive key search of 2^{32} keys and four pairs for 128 bits without an exhaustive key search. Our result shows the minimum number of pairs and that of fault injection points than the previous attacks. It takes about 2 seconds to retrieve 128 bits with four pairs on the normal PC. With two faults it takes about 8 minutes to find 128 bits.

The general countermeasure against DPA on AES state is to recompute the last three rounds of AES and compare it with the original output. However, if the key schedule is not re-done for the re-computation of the last three rounds it cannot prevent DPA on AES key schedule. Therefore we can conclude that key scheduling process as well as encryption process should be protected against fault attacks.

References

1. National institute of standards and technology, Advanced Encryption Standards. NIST FIPS PUB 197 (2001)
2. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
3. Blömer, J., Seifert, J.-P.: Fault based cryptanalysis of the advanced encryption standard (AES). In: Wright, R.N. (ed.) FC 2003. LNCS, vol. 2742, pp. 162–181. Springer, Heidelberg (2003)
4. Boneh, D., DeMillo, R., Lipton, R.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
5. Boneh, D., DeMillo, R., Lipton, R.: On the importance of eliminating errors in cryptographic computations. Journal of Cryptology 14(2), 101–119 (2001); An earlier version appears in [4]

6. Chen, C.-N., Yen, S.-M.: Differential fault analysis on AES key schedule and some countermeasures. In: Safavi-Naini, R., Seberry, J. (eds.) ACISP 2003. LNCS, vol. 2727, Springer, Heidelberg (2003)
7. Dusart, P., Letourneux, G., Vivolo, O.: Differential fault analysis on A.E.S (2003)/10, http://eprint.iacr.org/
8. Giraud, C.: DFA on AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2004. LNCS, vol. 3373, pp. 27–41. Springer, Heidelberg (2005)
9. Moradi, A., Shalmani, M.T.M., Salmasizadeh, M.: A generalized method of differential fault attack against AES cryptosystem. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 91–100. Springer, Heidelberg (2006)
10. Peacham, D., Thomas, B.: A DFA attack against the AES key schedule. SiVenture White Paper 001 (26 October 2006), http://www.siventure.com/pdfs/AES_KeySchedule_DFA_whitepaper.pdf
11. Piret, G., Quisquater, J.-J.: A differential fault attack technique against SPN structures, with application to AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 77–88. Springer, Heidelberg (2003)
12. Takahashi, J., Fukunaga, T.: Differential fault analysis on the AES key schedule. IACR Eprint archive 2007-480
13. Takahashi, J., Fukunaga, T., Yamakoshi, K.: DFA mechanism on the AES key schedule. In: Proc. of the Fourth International Workshop, FDTC 2007, pp. 62–72 (2007)