DFA on AES

Christophe Giraud

Oberthur Card Systems,
25, rue Auguste Blanche, 92800 Puteaux, France.
c.giraud@oberthurcs.com

Abstract. In this paper we describe two different DFA attacks on the AES. The first one uses a fault model that induces a fault on only one bit of an intermediate result, hence allowing us to obtain the key by using 50 faulty ciphertexts for an AES-128. The second attack uses a more realistic fault model: we assume that we may induce a fault on a whole byte. For an AES-128, this second attack provides the key by using less than 250 faulty ciphertexts. Moreover, this attack has been successfully put into practice on a smart card.

Keywords: AES, DFA, side-channel attacks, smartcards.

1 Introduction

Since Boneh, Demillo and Lipton introduced a cryptanalytic attack in September 1996 based on the fact that errors may be induced on smartcards during the computation of a cryptographic algorithm to find the key [8, 9], many papers have been published on this subject. Boneh *et al.* succeeded in breaking an RSA CRT with both a correct and a faulty signature of the same message. Lenstra then improved their attack [15] by finding one of the factors of the public modulus using only one faulty signature of a known message. In October 1996, Biham and Shamir published an attack on secret key cryptosystems [6] entitled Differential Fault Analysis (DFA). In 2000, Biehl, Meyer and Müller presented a paper describing two types of DFA attacks on elliptic curve cryptosystems [5] which were later refined by Ciet and Joye [10].

DFA is frequently used nowadays to test the security of cryptographic smartcards applications, especially those using the DES. On the 2$^{\text{nd}}$ October 2000, the AES was chosen to be the successor of the DES and, since then, it is used more and more in smartcards applications. So it seems interesting to investigate what is feasible on the AES by using DFA. Unfortunately, the DFA attack on symmetric cryptosystems proposed by Biham and Shamir [6] does not work on the AES. This is why we work to find a way to attack the AES by using DFA.

On a smartcard, a fault may be induced by its owner in many ways, such as power glitch, clock pulse or radiation of many kinds (laser, etc...). These external interventions may induce a fault, but we do not know the real impact on the computation inside the card. This is why, in this paper, we use two types of fault models. The first fault model assumes that the fault occurs on only one bit of a temporary result. Of course such a fault may be difficult to induce in practice, so the second fault model assumes that the induced fault may change a whole byte. The first fault model is the same as the one used in [5, 6, 8, 9] and was put into practice in 2002 by Skorobogatov and Anderson [18].

In the course of this paper, we describe the AES algorithm before looking at a DFA attack on the AES by using our first fault model. This attack allows us to find the AES-128 key by using 50 faulty ciphertexts. We then explain a more practical DFA attack on an AES-128 by using our second fault model. This attack allows us to find the key by using less than 250 faulty ciphertexts. Finally we present the second attack on a real smart card from a practical point of view. To the best of our knowledge, it is the first time an attack on the AES by DFA has been successfully put into practice.

2 AES

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

- we denote by M the plaintext and by K the AES key,
- M^i denotes the temporary cipher result after the i^{th} round and M^i_j the j^{th} byte of M^i,
- K^i denotes the i^{th} AES round key and K^i_j the j^{th} byte of K^i,
- C denotes the correct ciphertext and C_j the j^{th} byte of C,
- D denotes a faulty ciphertext and D_j the j^{th} byte of D.

The following section gives a general description of the AES. For more information, the reader can refer to [17, 11].

2.1 General description

The AES algorithm is capable of encrypting or decrypting data blocks of 128 bits by using cryptographic keys of 128, 192 or 256 bits.

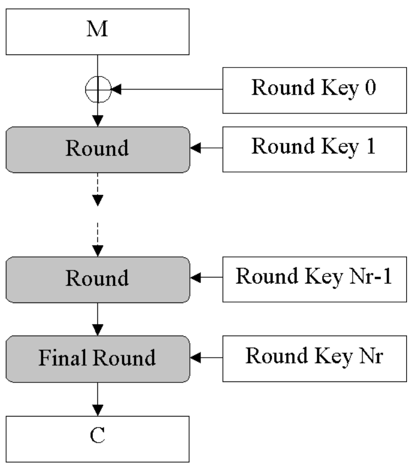

Fig. 1. General structure of AES.

The AES key scheduling provides $N_r + 1$ round keys. The number of rounds N_r is dependent on the key length as shown in the following table:

	Key length	Number of Rounds
AES-128	128	10
AES-192	192	12
AES-256	256	14

A 16-byte temporary result is represented as a two-dimensional array of bytes consisting of 4 rows and 4 columns. For example, $M^i = (M_0^i, ..., M_{15}^i)$ is represented by the following array:

M_0^i	M_4^i	M_8^i	M_{12}^i
M_1^i	M_5^i	M_9^i	M_{13}^i
M_2^i	M_6^i	M_{10}^i	M_{14}^i
M_3^i	M_7^i	M_{11}^i	M_{15}^i

2.2 A round

The Round function is composed of 4 transformations: SubBytes (SB), ShiftRows (SR), MixColumns (MC) and a bit-per-bit XOR with a round key. The Final Round of the AES is composed of the same functions as a classical Round except that it does not include the MixColumns transformation.

SubBytes This transformation is a non-linear byte substitution and operates on each input byte independently. So, we apply the substitution table (S-box) on each byte of the input to obtain the output.

ShiftRows The rows of the temporary result are cyclically shifted over different offsets. Row 0 is not shifted, Row 1 is shifted over 1 byte, Row 2 is shifted over 2 bytes and Row 3 is shifted over 3 bytes.

MixColumns Here, the columns of the temporary result are considered as polynomials over $\mathbb{F}_{2^8}$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $a(x) = 03 * x^3 + 01 * x^2 + 01 * x + 02$.

Notice that if we change a byte of the input of SubBytes or of ShiftRows, it will change one byte of the output. But for the MixColumns transformation, changing a byte of the input induces a modification of four output bytes.

2.3 Key Scheduling

The Key Scheduling generates the round keys from the AES key K by using 2 functions: the Key Expansion and the Round Key Selection.

Key Expansion This function computes from the AES key, an expanded key of length equal to the message block length multiplied by the number of rounds plus 1.
The expanded key is a linear array of 4-byte words and is denoted by $EK[4*(N_r+1)]$ where N_r is the number of rounds. If we denote by N_k the key length in words, the key expansion is described in the following pseudo code:

```
KeyExpansion(byte Key[4 * N_k], word EK[4 * (N_r + 1)])
{
  word temp;
  for (i = 0 ; i < N_k ; i++)
     EK[i] = (Key[4 * i], Key[4 * i + 1], Key[4 * i + 2], Key[4 * i + 3]);
  for (i = N_k ; i < 4 * (N_r + 1) ; i++)
     temp = EK[i − 1];
     if (i mod N_k = 0)
        temp = SubWord(RotWord(temp)) ⊕ Rcon[i/N_k];
     else if ((N_k > 6) and (i mod N_k = 4))
        temp = SubWord(temp);
     EK[i] = EK[i − N_k] ⊕ temp;
}
```

where:

- $SubWord()$ is a function that applies the AES S-box at each byte of the 4-byte input to produce an output word,
- $RotWord()$ is a cyclic rotation such that a 4-byte input (a, b, c, d) produces the 4-byte output (b, c, d, a),
- the round constant word array, $Rcon[i]$, is defined by $Rcon[i] = (x^{i-1}, \{00\}, \{00\}, \{00\})$ with x^{i-1} being powers of x (x is denoted as $\{02\}$) in the field $\mathbb{F}_{2^8}$.

Round Key Selection This routine extracts the 128-bit round keys from the Expanded Key.

Example of Key Scheduling for an AES-128

AES Key:	K								
Expanded Key:	EK_0	EK_1	EK_2	EK_3	EK_4	EK_5	EK_6	EK_7	...
Round Keys:	Round Key 0				Round Key 1				...

where

- $(EK_0, ..., EK_3)$ is the 128-bit AES key K,
- $EK_4 = EK_0 \oplus SubWord(RotWord(EK_3)) \oplus Rcon[1]$,
- $EK_5 = EK_1 \oplus EK_4$,
- $EK_6 = EK_2 \oplus EK_5$,
- $EK_7 = EK_3 \oplus EK_6$, ...

3 Bit-fault attack

In this section, by using a DFA attack where a fault occurs on only one bit of the temporary cipher result at the beginning of the Final Round, we show how to obtain the entire last round key, i.e. the AES key for an AES-128. For more information about this fault model, the reader can refer to [18].

For the sake of simplicity, we describe the attack on an AES using a 128-bit key.

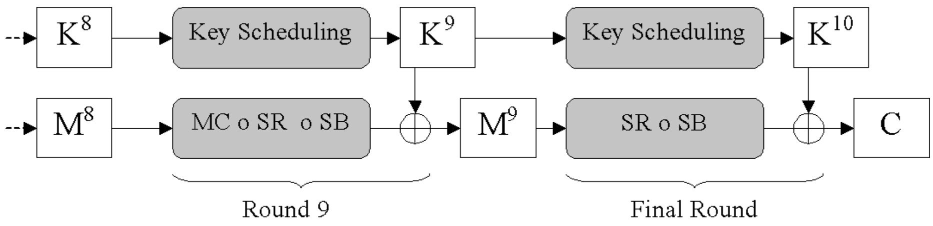

Fig. 2. The last rounds of an AES-128.

By definition, we have

$$C = ShiftRows(SubBytes(M^9)) \oplus K^{10} \tag{1}$$

Let us denote by $SubByte(M_j^i)$ the result of the substitution table applied on the byte M_j^i and by $ShiftRow(j)$ the position of the j^{th} byte of a temporary result after applying the $ShiftRows$ transformation.
So, we have from (1)

$$C_{ShiftRow(i)} = SubByte(M_i^9) \oplus K^{10}_{ShiftRow(i)}, \quad \forall i \in \{0, ..., 15\} \tag{2}$$

If we induce a fault e_j on one bit of the j^{th} byte of the temporary cipher result M^9 just before the Final Round, we obtain a faulty ciphertext D where:

$$D_{ShiftRow(j)} = SubByte(M_j^9 \oplus e_j) \oplus K^{10}_{ShiftRow(j)} \tag{3}$$

and for all $i \in \{0, ..., 15\} \backslash \{j\}$, we have:

$$D_{ShiftRow(i)} = SubByte(M_i^9) \oplus K^{10}_{ShiftRow(i)} \tag{4}$$

So, if there is no induced fault on the i^{th} byte of M^9, we obtain from (2) and (4)

$$C_{ShiftRow(i)} \oplus D_{ShiftRow(i)} = 0 \tag{5}$$

and if there is an induced fault on M_j^9, we have from (2) and (3)

$$C_{ShiftRow(j)} \oplus D_{ShiftRow(j)} = SubByte(M_j^9) \oplus SubByte(M_j^9 \oplus e_j) \tag{6}$$

Firstly, we determine $ShiftRow(j)$ which is the position of the only non-zero byte of $C \oplus D$ and we thus obtain j. We then use a counting method in order to find M_j^9: we guess the single bit fault e_j and we find a set of possible values for M_j^9 which verify (6). For each of these values, we increase the corresponding counter by 1. With another faulty ciphertext, the right value for M_j^9 is expected to be counted more frequently than any wrong value, and can thus be identified. Then we iterate the previous process to obtain all the other bytes of M^9.

Now, as we know the value of the ciphertext C and the value of M^9, we can easily obtain the last round key K^{10} from the formula (1) and consequently the AES key K by applying the inverse of the Key Scheduling to K^{10}.

By using 3 faulty ciphertexts with faults induced on the same byte of M^9, we have a 97% chance of success to obtain this byte (cf. appendice A). So, it is possible to obtain the 128-bit AES key by using less than 50 faulty ciphertexts.

This attack operates independently on each byte, so if we succeed in inducing a fault on only one bit on several bytes of M^9, we reduce the number of faulty ciphertexts required to obtain the key.

We notice that this attack also operates on the AES-192 and on the AES-256. In such cases, we obtain the last round key, i.e. the security of the AES-192 is reduced from 24 to 8 bytes and the security of the AES-256 is reduced from 32 to 16 bytes.

This attack is powerful but requires inducing a fault on only one bit at the time of a precise event (i.e. at the beginning of the last round) which may be difficult in practice.

4 A second type of DFA attack on the AES-128

This DFA attack uses the fault model based on inducing a fault which may change a whole byte of a temporary result. This attack, which only works on an AES using a 128-bit key, is divided into 3 steps :

1. we obtain the last 4 bytes of K^9 by exploiting the faulty ciphertexts obtained when a fault is introduced on K^9, just before the computation of K^{10},
2. we obtain another 4 bytes of K^9 by exploiting the faulty ciphertexts obtained when a fault is introduced on K^8, just before the computation of K^9,
3. finally, we obtain the AES key K by exploiting the faulty ciphertexts obtained by introducing a fault on M^8 before entering Round 9 and by using the 8 bytes of K^9 disclosed in steps 1 and 2.

In smartcard implementations, each round key is computed on-the-fly. In the following section, "attack on K^i" means that the correct i^{th} round key has been used for the cipher and that a fault has been induced on this round key before computing the $i+1^{\text{th}}$ round key which is a faulty round key.

4.1 DFA attack on K^9

We suppose that we know both the correct ciphertext C and a faulty ciphertext D of the same plaintext M and that the fault occurs on one of the bytes of K^9 just

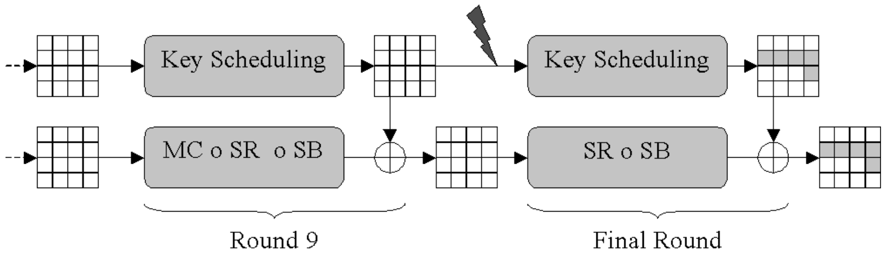

Fig. 3. Fault on the 14$^{\text{th}}$ byte of the penultimate round key K^9.

before computing K^{10} as shown in figure 3, where the shaded squares represent the bytes affected by the fault.
We want the fault to occur on one of the last 4 bytes of K^9. In that case, two of the last 4 bytes of the faulty ciphertext will be different from those of the correct ciphertext. We must hence check if this condition is true: if it is not, we abandon this faulty ciphertext and we generate another faulty ciphertext with a fault on K^9 and we test it again.

Now, we will see that it is possible to identify:

- the position j of the byte on which the fault occurred
- and the value e_j of this fault.

If we suppose that a fault e_j occurs on the j^{th} byte of K^9 $(12 \leq j \leq 15)$ just before the Final Round, there will only be one non-zero byte in the first 4 bytes of $C \oplus D$. If we denote this byte the k^{th} $(0 \leq k \leq 3)$, j is then defined by

$$j = (k + 1 \bmod 4) + 12 \tag{7}$$

By computing $C \oplus D$, we determine k and thus obtain j.

By definition, we have:

$$\forall i \in \{0, ..., 15\}, \quad C_i = SubByte(M^9_{ShiftRow^{-1}(i)}) \oplus K^{10}_i \tag{8}$$

More precisely:

- if $i = 0$:

$$C_i = SubByte(M^9_{ShiftRow^{-1}(i)}) \oplus SubByte(K^9_{(i+1 \ mod \ 4)+12}) \oplus K^9_i \oplus 0\text{x}36 \tag{9}$$

- if $i \in \{1, 2, 3\}$:

$$C_i = SubByte(M^9_{ShiftRow^{-1}(i)}) \oplus SubByte(K^9_{(i+1 \ mod \ 4)+12}) \oplus K^9_i \tag{10}$$

We also have for the faulty ciphertext:

$$D_j = SubByte(M^9_{ShiftRow^{-1}(j)}) \oplus K^{10}_j \oplus e_j \tag{11}$$

and

- if $k = 0$:

$$D_k = SubByte(M^9_{ShiftRow^{-1}(k)}) \oplus SubByte(K^9_j \oplus e_j) \oplus K^9_k \oplus 0\text{x}36 \tag{12}$$

- if $k \in \{1, 2, 3\}$:

$$D_k = SubByte(M^9_{ShiftRow^{-1}(k)}) \oplus SubByte(K^9_j \oplus e_j) \oplus K^9_k \qquad (13)$$

It is easy to see, from (8) and (11), that the value of the fault e_j is equal to $C_j \oplus D_j$.

We have now identify the position j of the byte on which the fault occurred and the value e_j of this fault. Let us see how to use this information to obtain the value of K^9_j.

From (9), (10), (12) and (13), we have the equation

$$C_k \oplus D_k = SubByte(K^9_j) \oplus SubByte(K^9_j \oplus e_j) \qquad (14)$$

We know the value of $C_k \oplus D_k$ and the value of e_j. So, we search the possible values $x \in \{0, ..., 255\}$ which satisfy the equation

$$C_k \oplus D_k = SubByte(x) \oplus SubByte(x \oplus e_j) \qquad (15)$$

We obtain K^9_j and $K^9_j \oplus e_j$ as solutions to (15). So, if we obtain another faulty ciphertext with a fault e'_j ($e'_j \neq e_j$) which occurs on the same byte j of K^9, we obtain K^9_j and $K^9_j \oplus e'_j$ as solutions. This allows us to deduce the value of K^9_j because it is the only value that appears in both solution.

With this attack, we obtain the values of the last 4 bytes (K^9_{12} to K^9_{15}) of the round key K^9 with 32 faulty ciphertexts on average.

4.2 Attack on K^8

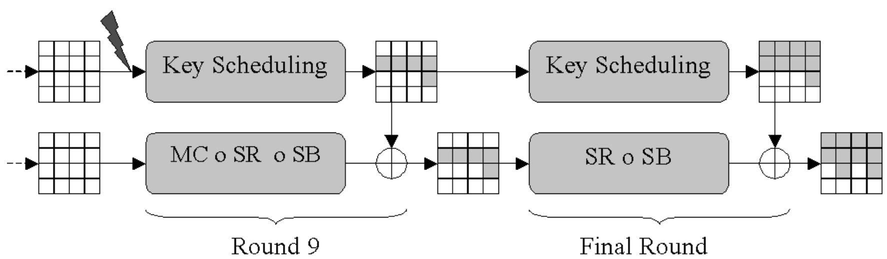

Fig. 4. Fault on the 14^{th} byte of the antepenultimate round key K^8.

Now, we will see how to obtain the 4 bytes K^9_8 to K^9_{11}. We use faulty ciphertexts obtained when the fault e_j occurred on one byte of K^8 (lets say the j^{th} byte) before Round 9.

We want the fault to occur on one of the last 4 bytes of K^8. If it is the case, there will only be one zero byte in the last 4 bytes of $C \oplus D$. So we test this condition and if it is false, we generate another faulty ciphertext with a fault induced on K^8 and we test it again.

As in section 4.1, we will:

– identify the position j of the byte on which the fault occurred
– and obtain the value e_j of this fault.

If we denote by l the position of the zero byte in the last 4 bytes of $C \oplus D$ ($12 \leq l \leq 15$), j is then defined by

$$j = (l - 1 \bmod 4) + 12 \tag{16}$$

Now, we know on which byte of K^8 the fault occurred.

We have, for the faulty ciphertext D:

$$\begin{cases} D_j = SubByte(M^9_{ShiftRow^{-1}(j)}) \oplus K^{10}_j \oplus e_j & \text{if } j \neq 12 \\ D_j = SubByte(M^9_{ShiftRow^{-1}(j)} \oplus e_j) \oplus K^{10}_j \oplus e_j & \text{if } j = 12 \end{cases} \tag{17}$$

and for the correct ciphertext:

$$C_i = SubByte(M^9_{ShiftRow^{-1}(i)}) \oplus K^{10}_i \quad \forall i \in \{0, ..., 15\} \tag{18}$$

– If $j \neq 12$, we easily obtain the value of e_j which is equal to $C_j \oplus D_j$.
– But, if $j = 12$, the $ShiftRows$ transformation does not affect the 12^{th} byte and we cannot directly obtain the value of the fault e_j. We only know that

$$C_j \oplus D_j = SubByte(a) \oplus SubByte(a \oplus e_j) \oplus e_j \tag{19}$$

for a certain 8-bit value a. In this case, we guess the fault e_j and we look for a value a which satisfies (19). If such a value exists, we assume that our guess may be correct and we keep it as a possible value for the fault e_j. We obtain between 107 and 146 different possible values for e_j depending on the value of $C_j \oplus D_j$; the average is about 127.

Now, we have identify the position j of the byte on which the fault occurred and the value e_j of this fault if $j \neq 12$ or a set of possible values if $j = 12$. Let us see how to use this information to obtain the value of K^8_j.

If we induce a fault on K^8_j ($12 \leq j \leq 15$), the 4 bytes of the faulty 9^{th} round key at position $(j - 1 \bmod 12) + 4n$, $n \in \{0, 1, 2, 3\}$, are different from the bytes at the same position of the correct 9^{th} round key K^9. These four differences between the correct and the faulty 9^{th} round key are equal and we denote this difference f_j.
If we denote $k = (j - 1 \bmod 4) + 12$, we have $K^9_k \oplus f_j$ as the value of the k^{th} byte of the faulty 9^{th} round key.
So, we have:

- if $j = 14$:

$$\begin{aligned} D_{j-2 \bmod 4} = {} & SubByte(M^9_{ShiftRow^{-1}(j-2 \bmod 4)}) \oplus SubByte(K^9_k \oplus f_j) \\ & \oplus K^9_{j-2 \bmod 4} \oplus 0\text{x}36 \end{aligned} \tag{20}$$

- if $j \in \{12, 13, 15\}$:

$$\begin{aligned} D_{j-2 \bmod 4} = {} & SubByte(M^9_{ShiftRow^{-1}(j-2 \bmod 4)}) \oplus SubByte(K^9_k \oplus f_j) \\ & \oplus K^9_{j-2 \bmod 4} \end{aligned} \tag{21}$$

And we obtain from (9), (10), (20) and (21):

$$C_{j-2 \bmod 4} \oplus D_{j-2 \bmod 4} = SubByte(K_k^9) \oplus SubByte(K_k^9 \oplus f_j) \tag{22}$$

As we know the value of K_k^9 from the previous attack (section 4.1), we can easily find the value of f_j which satisfies (22).

Moreover, $K_{j-1 \bmod 4}^9 \oplus f_j$ is the value of the $(j-1 \bmod 12)^{\text{th}}$ byte of the faulty 9^{th} round key. So, we have for the faulty Key Scheduling:

- if $j = 13$:

$$SubByte(K_j^8 \oplus e_j) \oplus K_{j-1 \bmod 4}^8 \oplus 0\text{x}36 = K_{j-1 \bmod 4}^9 \oplus f_j \tag{23}$$

- if $j \in \{12, 14, 15\}$:

$$SubByte(K_j^8 \oplus e_j) \oplus K_{j-1 \bmod 4}^8 = K_{j-1 \bmod 4}^9 \oplus f_j \tag{24}$$

and for the correct Key Scheduling:

- if $j = 13$:

$$SubByte(K_j^8) \oplus K_{j-1 \bmod 4}^8 \oplus 0\text{x}36 = K_{j-1 \bmod 4}^9 \tag{25}$$

- if $j \in \{12, 14, 15\}$:

$$SubByte(K_j^8) \oplus K_{j-1 \bmod 4}^8 = K_{j-1 \bmod 4}^9 \tag{26}$$

We obtain from (23), (24), (25) and (26):

$$f_j = SubByte(K_j^8 \oplus e_j) \oplus SubByte(K_j^8) \tag{27}$$

With the value of f_j previously obtained from (22), we find all the possible values K_j^8 which satisfy (27).
As in section 3, we use a counting method in order to find the correct K_j^8. The right K_j^8 can be obtained quickly when $j \neq 12$ because we know the value of the fault e_j. However, if $j = 12$ it is more difficult because there are many possible values for e_j (between 107 and 146). Although we need more faulty ciphertexts to determine K_{12}^8 than to determine K_{13}^8, K_{14}^8 or K_{15}^8, the number required is relatively low. We need approximately 13 faulty ciphertexts from the same plaintext to obtain K_{12}^8 and only 2 to obtain K_{13}^8, K_{14}^8 or K_{15}^8 (by using simulation, we find that we have a 90% chance of success to determine K_{12}^8 if we use 10 faulty ciphertexts and this percentage grows up to 99% if we use 13 faulty ciphertexts).
Finally, to obtain K_8^9, K_9^9, K_{10}^9 and K_{11}^9, we use the following formula:

$$K_i^9 = K_{i+4}^8 \oplus K_{i+4}^9 \quad \forall i \in \{8, ..., 11\} \tag{28}$$

At this step, we have obtained the last 8 bytes of the penultimate round key K^9 by using about 240 faulty ciphertexts.

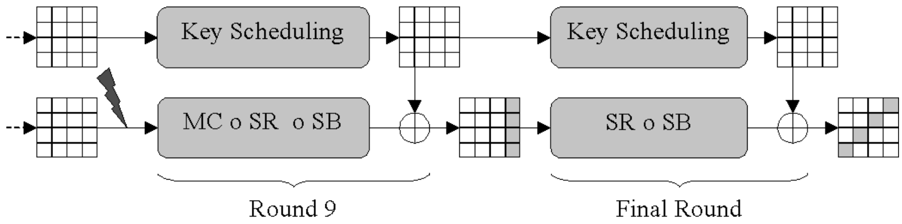

Fig. 5. Fault on the 11^{th} byte of M^8.

4.3 DFA attack on M^8

Before entering Round 9, we assume that a fault on one byte of M^8 has been induced. As we have determined the last 8 bytes of K^9, we want the fault to occur on a byte of M^8 which will be XORed with one of the last 8 bytes of K^9 after $MC \circ SR \circ SB$. Due to the $ShiftRows$ and $MixColumns$ transformations, we know that if we induce a fault on M^8_{12}, M^8_1, M^8_6 or on M^8_{11} (resp. on M^8_8, M^8_{13}, M^8_2 or on M^8_7), the result of these bytes after $MC \circ SR \circ SB$ will be XORed with K^9_{12} to K^9_{15} (resp. K^9_8 to K^9_{11}). So, we want a fault to occur on one of these 8 bytes of M^8 and to test if this happens, we look at the faulty ciphertext: if only the 4 bytes (D_{12}, D_9, D_6, D_3) (resp. (D_8, D_5, D_2, D_{15})) differ from (C_{12}, C_9, C_6, C_3) (resp. (C_8, C_5, C_2, C_{15})) of the correct ciphertext, this shows that the fault occurred on one of the 4 bytes (M^8_{12}, M^8_1, M^8_6, M^8_{11}) (resp. (M^8_8, M^8_{13}, M^8_2, M^8_7)).

In the following, let (D_{12}, D_9, D_6, D_3) be different from (C_{12}, C_9, C_6, C_3).
We guess the fault e_j ($1 \leq e_j \leq 255$) and we list all the 4-byte values V which verify one of the following equations:

$$\begin{array}{l} SB(MC(V) \oplus K^9_{12-15}) \oplus SB(MC(V \oplus (0,0,0,e_j)) \oplus K^9_{12-15}) = TR_{12-15} \\ SB(MC(V) \oplus K^9_{12-15}) \oplus SB(MC(V \oplus (0,0,e_j,0)) \oplus K^9_{12-15}) = TR_{12-15} \\ SB(MC(V) \oplus K^9_{12-15}) \oplus SB(MC(V \oplus (0,e_j,0,0)) \oplus K^9_{12-15}) = TR_{12-15} \\ SB(MC(V) \oplus K^9_{12-15}) \oplus SB(MC(V \oplus (e_j,0,0,0)) \oplus K^9_{12-15}) = TR_{12-15} \end{array} \tag{29}$$

where K^9_{12-15} denotes the 4-byte value $(K^9_{12}, K^9_{13}, K^9_{14}, K^9_{15})$ and TR_{12-15} the 4-byte value $(C \oplus D)_{ShiftRow(12-15)} = (C_{12} \oplus D_{12}, C_9 \oplus D_9, C_6 \oplus D_6, C_3 \oplus D_3)$.

So, if we apply the same reasoning to another faulty ciphertext which differs from the correct ciphertext on (D_{12}, D_9, D_6, D_3), we obtain another list of 4-byte values. There will only be one 4-byte value present in both lists and this will be the correct value of the last 4 bytes of the temporary result before the $MixColumns$ transformation in Round 9.

Proceeding in the same way with two different faulty ciphertexts in which (D_8, D_5, D_2, D_{15}) differ from (C_8, C_5, C_2, C_{15}), we obtain the correct 8^{th} to 11^{th} bytes of the temporary result before the $MixColumns$ transformation in Round 9.
The previous exhaustive searches on 4-byte values require some time and it takes about 5 days to obtain these 8 bytes on a 1 GHz Pentium.
Having now identified the last 8 bytes of the temporary cipher result before the

MixColumns transformation in Round 9, we apply *MixColumns* to these 8 bytes. We then XOR the result with the corresponding bytes of K^9 (i.e. K^9_8 to K^9_{15}) and we apply $SR \circ SB$. This result is a part of the correct temporary result before the XOR with K^{10}. So, we XOR it with the corresponding bytes of the ciphertext C to obtain the bytes K^{10}_2, K^{10}_3, K^{10}_5, K^{10}_6, K^{10}_8, K^{10}_9, K^{10}_{12} and K^{10}_{15}. Using the known bytes of K^9, we obtain 6 other bytes of K^{10} by the following relations:

$$\begin{array}{l} K^{10}_{13} = K^{10}_9 \oplus K^9_{13} \\ K^{10}_{11} = K^{10}_{15} \oplus K^9_{15} \\ K^{10}_{10} = K^{10}_6 \oplus K^9_{10} \\ K^{10}_{14} = K^{10}_{10} \oplus K^9_{14} \\ K^{10}_7 = K^{10}_{11} \oplus K^9_{11} \\ K^{10}_4 = K^{10}_8 \oplus K^9_8 \end{array} \tag{30}$$

Finally, we find the last 2 unknown bytes of K^{10} by a very fast exhaustive search and we obtain the AES key from K^{10} by applying the inverse of the Key Scheduling.
Theoretically, we obtain the full AES key by using less than 250 faulty ciphertexts.

5 In practice

We implemented the algorithmic part of the second attack on an AES-128 and, by simulating faults on random bytes of K8, K9 and M9, we found the whole AES key by using 250 faulty ciphertexts. This was easily done on a computer but we were yet to discover if our second attack could be successfully put into practice on a smart card.

By using a microscope, a modified camera flash and a computer, we attacked an AES-128 on an 8-bit smart card (to make the attack easier, we used a known AES code). Firstly, we had to find out where the light flash was most efficient on the surface of the chip and then we had to synchronize the flash with the operations we wanted to disturb.
We even succeeded in inducing a fault for nearly every execution of the AES, we needed a lot of tries to obtain a “good” faulty ciphertext. Indeed, most of the time, the induced fault affected 4 or 8 bytes of the temporary result.
To recover the key, we needed numerous tries: more than 1000 AES executions were required.
If we had had a laser we could have shortened the length of the flash and hence obtained a “good” faulty ciphertext more frequently by disturbing the chip for a very short time, i.e. during the treatment of only one byte.
This experience demonstrates that AES on smart cards must now be implemented not only with SPA/DPA countermeasures but also with DFA countermeasures.

6 Conclusion

Although DFA on the DES is a well-known attack, it is impossible to directly apply Biham and Shamir’s attack to the AES as the latter does not have the Feistel Structure.

This paper extends the operative field of differential fault attacks by describing how to perform two different DFA attacks on the AES. Each of these attacks allow us to obtain the full AES key in the case of a 128-bit key length. We note that it is possible to put the second attack into practice on smart cards. However, it is easy to avoid both attacks. For exemple, this can be done by doubling the last two rounds and by checking if the two outputs are equal.

Acknowledgments

I would like to thank Mathieu Ciet for his valuable comments as well as Erik Knudsen for many helpful discussions. The practical attack would never have been possible without the help of Hugues Thiebeauld. And the last but not the least, I am really grateful to Julia Bradley for her help and support during the writing of this paper.

References

1. R. Anderson and M. Kuhn, *Tamper Resistance - a cautionary note*, proceedings of 2nd USENIX Workshop on Electronic Commerce, pp. 1-11, 1996.
2. R. Anderson and M. Kuhn, *Low cost attacks on tamper resistant devices*, Springer, Lecture Notes in Computer Science vol. 1361, proceedings of 5th Security Protocols Workshop, pp. 125-136, 1997.
3. C. Aumüller, P. Bier, W. Fischer, P. Hofreiter and J.-P. Seifert, *Fault attacks on RSA with CRT: Concrete Results and Practical Countermeasures*, Springer, Lecture Notes in Computer Science vol. 2523, proceedings of CHES 2002, pp. 260-275, 2002.
4. F. Bao, R. Deng, Y. Han, A. Jeng, A. D. Narasimhalu and T.-H. Ngair, *Breaking Public Key Cryptosystems an Tamper Resistance Devices in the Presence of Transient Fault*, Springer, Lecture Notes in Computer Science vol. 1361, proceedings of 5th Security Protocols WorkShop, pp. 115-124, 1997.
5. I. Biehl, B. Meyer and V. Müller, *Differential Fault Analysis on Elliptic Curve Cryptosystems*, Springer Lecture Notes in Computer Science vol. 1880, Advances in Cryptology, proceedings of CRYPTO 2000, pp. 131-146, 2000.
6. E. Biham and A. Shamir, *Differential Fault Analysis of Secret Key Cryptosystems*, Springer, Lecture Notes in Computer Science vol. 1294, Advances in Cryptology, proceedings of CRYPTO'97, pp. 513-525, 1997.
7. J. Blömer and J.-P. Seifert, *Fault based cryptanalysis of the Advanced Encryption Standard (AES)*, Springer, Lecture Notes in Computer Science, proceedings of FC'03, 2003.
8. D. Boneh, R. A. Demillo and R. J. Lipton, *On the Importance of Checking Cryptographic Protocols for Faults*, Springer, Lecture Notes in Computer Science vol. 1233, Advances in Cryptology, proceedings of EUROCRYPT'97, pp. 37-51, 1997.
9. D. Boneh, R. A. DeMillo and R. J. Lipton, *On the Importance of Eliminating Errors in Cryptographic Computations*, Springer, Journal of Cryptology 14(2), pp. 101-119, 2001. An earlier version was published at EUROCRYPT'97 [8].
10. M. Ciet and M. Joye, *Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults*, IACR, Cryptology ePrint Archive, Available from `http://eprint.iacr.org/2003/028`, 2003.
11. J. Daemen and V. Rijmen, *The Design of Rijndael*, Springer-Verlag, 2002.
12. M. Joye, F. Koeune and J.-J. Quisquater, *Further results on Chinese remaindering*, UCL Technical Report CG-1997/1, Available from `http://www.dice.ucl.ac.be/crypto/techreports.html`, 1997.
13. M. Joye, A. K. Lenstra and J.-J. Quisquater, *Chinese Remaindering Based Cryptosystems in the Presence of Faults*, Springer, Journal of Cryptology 12(4), pp. 241-246, 1999.
14. M. Joye, J.-J. Quisquater, S.-M. Yen and M. Yung, *Observability Analysis - Detecting When Improved Cryptosystems Fail* -, Springer, Lecture Notes in Computer Science vol. 2271, Advances in Cryptology, proceedings of RSA Conference 2002, pp. 17-29, 2002.

15. A. K. Lenstra, *Memo on RSA Signature Generation in the Presence of Faults*, Manuscript, Available from the author at `arjen.lenstra@citicorp.com`, 1996.
16. D. P. Maher, *Fault Induction Attacks, Tamper Resistance, and Hostile Reverse Engineering in Perspective*, Springer, Lecture Notes in Computer Science vol. 1318, proceedings of Financial Cryptography '97, pp. 109-121, 1997.
17. National Institute of Standards and Technology, *Advanced Encryption Standard*, NIST FIPS PUB 197, 2001.
18. S. Skorobogatov and R. Anderson, *Optical Fault Induction Attack*, Springer, Lecture Notes in Computer Science vol. 2523, proceedings of CHES 2002, pp. 2-12, 2002.
19. S.-M. Yen and J. Z. Chen, *A DFA on Rijndael*, proceedings of ISC 2002, 2002.
20. S.-M. Yen and M. Joye, *Checking before output may not be enough against fault-based cryptanalysis*, IEEE Transactions on Computers 49(9), pp. 967-970, 2000.
21. S.-M. Yen, S.-J. Kim, S.-G. Lim and S.-J. Moon, *RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis*, Springer, Lecture Notes in Computer Science vol. 2288, proceedings of ICISC 2001, pp. 397-413, 2001.
22. S.-M. Yen, S.-J. Kim, S.-G. Lim and S.-J. Moon, *A Countermeasure against one Physical Cryptanalysis May Benefit Another Attack*, Springer, Lecture Notes in Computer Science vol. 2288, proceedings of ICISC 2001, pp. 414-427, 2001.

A The first attack in more details

If a message M is ciphered by using an AES-128 and if a one-bit fault e_j is induced on M_j^9, we obtain a faulty ciphertext D. We then have the following equation:

$$C_{ShiftRow(j)} \oplus D_{ShiftRow(j)} = SubByte(M_j^9) \oplus SubByte(M_j^9 \oplus e_j) \tag{31}$$

For each faulty ciphertext we perform 8.2^8 tests, i.e. for all values of x between 0 and 255 and for $e_j \in \{0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80\}$, we test if the following equality holds :

$$C_{ShiftRow(j)} \oplus D_{ShiftRow(j)} = SubByte(x) \oplus SubByte(x \oplus e_j) \tag{32}$$

There is no solution to (32) if $C_{ShiftRow(j)} \oplus D_{ShiftRow(j)} = 185$, so this value can be excluded right away. By consecutively fixing the left hand side of (32) with the 254 possible values $\{1, .., 255\} \backslash \{185\}$ and by testing all possible pairs (x, e_j), we find that the number of possible values for M_j^9 varies from 2 to 14; the average is about 8.

If we assume that we are in the worst case, then we obtain 14 possible values for M_j^9 for each faulty ciphertext.
If we obtain another faulty ciphertext with an induced fault on M_j^9 we obtain another set of possible values for M_j^9. In each set we have the correct value of M_j^9, so to identify this value the other 13 values must be different from each other.
If we denote by A the set of these 13 values obtained with the first faulty ciphertext and by B the set of the possible values obtained with the second faulty ciphertext

except the correct value of M_j^9, we can identify the correct value with probability :

$$\begin{aligned} P_2 &= P(A \cap B = \emptyset) \\ &= P(|A \cap B| = 0) \\ &= \frac{\binom{255}{13} * \binom{255-13}{13}}{\binom{255}{13}^2} \\ &\simeq 50\% \end{aligned} \tag{33}$$

With a third faulty ciphertext with an induced fault on M_j^9 we obtain yet another set of 14 possibles values for M_j^9. If we denote by C this set without the correct value of M_j^9, we can identify the correct value with probability :

$$\begin{aligned} P_3 &= P(A \cap B \cap C = \emptyset) \\ &= P(|A \cap B \cap C| = 0) \\ &= \textstyle\sum_{k=0}^{min\{|A|,|B|\}} P(|A \cap B| = k,\ |A \cap B \cap C| = 0) \\ &= \textstyle\sum_{k=0}^{13} P(|A \cap B| = k) * P(|A \cap B \cap C| = 0 \ / \ |A \cap B| = k) \\ &= \textstyle\sum_{k=0}^{13} \frac{\binom{255}{13} * \binom{13}{k} * \binom{255-13}{13-k}}{\binom{255}{13}^2} * \frac{\binom{255}{k} * \binom{255-k}{13}}{\binom{255}{k} * \binom{255}{13}} \\ &\simeq 97\% \end{aligned} \tag{34}$$