Meet-in-the-Middle and Impossible Differential Fault Analysis on AES

Patrick Derbez[1], Pierre-Alain Fouque[1], and Delphine Leresteux[2]

[1] École Normale Supérieure, 45 rue d'Ulm, F-75230 Paris CEDEX 05
[2] DGA Information Superiority, BP7, 35998 Rennes Armées
{patrick.derbez,pierre-alain.fouque}@ens.fr,
delphine.leresteux@dga.defense.gouv.fr

Abstract. Since the early work of Piret and Quisquater on fault attacks against AES at CHES 2003, many works have been devoted to reduce the number of faults and to improve the time complexity of this attack. This attack is very efficient as a single fault is injected on the third round before the end, and then it allows to recover the whole secret key in 2^{32} in time and memory. However, since this attack, it is an open problem to know if provoking a fault at a former round of the cipher allows to recover the key. Indeed, since two rounds of AES achieve a full diffusion and adding protections against fault attack decreases the performance, some countermeasures propose to protect only the three first and last rounds. In this paper, we give an answer to this problem by showing two practical cryptographic attacks on one round earlier of AES-128 and for all keysize variants. The first attack requires 10 faults and its complexity is around 2^{40} in time and memory, an improvement allows only 5 faults and its complexity in memory is reduced to 2^{24} while the second one requires either 1000 or 45 faults depending on fault model and recovers the secret key in around 2^{40} in time and memory.

Keywords: AES, Differential Fault Analysis, Fault Attack, Impossible Differential Attack, Meet-in-the-Middle Attack.

1 Introduction

Fault Analysis was introduced in 1996 by Boneh *et al.* [8] against RSA-CRT implementations and soon after Biham and Shamir described differential fault attack on the DES block cipher [4]. Several techniques are known today to provoke faults during computations such as provoking a spike on the power supply, a glitch on the clock, or using external methods based on laser, Focused Ion Beam, or electromagnetic radiations [18]. These techniques usually target hardware or software components of smartcards, such as memory, register, data or address bus, assembly commands and so on [1]. After a query phase where the adversary collects pairs of correct and faulty ciphertexts, a cryptographic analysis of these data allows to reveal the secret key. The knowledge of a small difference at an inner computational step allows to *reduce* the analysis to a small number

B. Preneel and T. Takagi (Eds.): CHES 2011, LNCS 6917, pp. 274–291, 2011.
© International Association for Cryptologic Research 2011

of rounds of a block cipher for instance. On the AES block cipher, many such attacks have been proposed [6,14,17,23,27] and the first non trivial and the most efficient attack has been described by Piret and Quisquater in [27].

Related Works. The embedded software and hardware AES implementations are particularly vulnerable to side channel analysis [5,7,30]. Considering fault analysis, it exists actually three different categories of attacks. The first category is non cryptographic and allows to reduce the number of rounds by provoking a fault on the round counter [1,11]. In the second category, cryptographic attacks perform fault in the state during a round [6,14,17,23,27] and in the third category, the faults are performed during the key schedule [10,17,31].

Several fault models have been considered to attack AES implementations. The first one and the less common is the random bit fault [6], where a fault allows to switch a specific bit. The more realistic and widespread fault model is the random byte fault model used in the Piret-Quisquater attack [27], where a byte somewhere in the state is modified. These different fault models depend on the technique used to provoke the faults.

Piret and Quisquater described a general Differential Fault Analysis (DFA), against Substitution Permutation Network schemes in [27]. Their attack uses a single random byte fault model injected between the two last MixColumns of AES-128. They exploited only 2 pairs of correct and faulty ciphertexts. Since this article was published in 2003, many works have proposed to reduce the number of faults needed in [24,32], or to apply this attack to AES-192 and to AES-256 [20].

There exist two kinds of countermeasures to protect AES implementations against fault attacks. The first category detects fault injection with hardware sensors for instance. However, they are specifically designed for one precise fault injection mean and do not protect against all different fault injection techniques. The second one protects hardware implementation against fault effects. This kind of countermeasures increases the hardware surface requirement as well as the number of operations. As a consequence, there is a tradeoff between the protection and the efficiency and countermeasures essentially only protect from existing fault attacks by taking into account the known state-of-the-art fault analysis. Therefore, the first three and the last three rounds used to be protected [12]. The same kind of countermeasures has been performed on DES implementation and a rich literature has been devoted to increase the number of attacked rounds as it is done in [28]. Securing AES implementation consists in duplicating rounds, verifying operation with inverse operation for non-linear operations and with complementary property for linear ones, for example. Moreover, another approach computes and associates to each vulnerable intermediate value a cyclic redundancy checksum or, an error detection or correction code, for instance fault detection for AES S-Boxes [19] as it has been proposed at CHES 2008. Our attacks could target any operation between MixColumns at the 6^{th} round and MixColumns at the 7^{th} round. Another countermeasure consists in preventing from fault attack inside round [29]. However, it is possible to perform fault injection between rounds.

Our Results. We show that it is possible to mount realistic attacks between MixColumns at the 6^{th} round and MixColumns at the 7^{th} round on AES-128. In particular, we present one new attack and improve a second one at the 7^{th} round on AES-128. We mount our attacks in two different fault models. The first attack corresponds of a strong adversary who could choose or know the attacked byte at the chosen round. The cryptographic analysis relies on a meet-in-the-middle and its complexity is around 2^{42} in time and memory. It only requires 10 pairs of correct and faulty ciphertexts. Recently, in [9], authors developed automatic tool that allows us to automatically recover an improved attack with only 5 pairs and 2^{24} in memory. The second attack describes an adversary that targets any byte among 16 bytes of the inner state at the targeted round. It uses ideas similar to impossible differential attack and allows to recover the secret key using around 2^{40} time. However, this attack requires 1000 pairs. If the position is fixed, the number of faults is reduced to 45. We have verified this attack experimentally using glitch fault on the clock on an embedded microprocessor board which contains an AES software and simulated these two last attacks. Finally, we extend all the attacks to AES-192 and AES-256.

Table 1. Summary of Differential Fault Analysis presented in this paper

Attack	Section	Fault model	# of faults	AES-128 cost	AES-192 & AES-256 cost
Meet-in-the-Middle	3.2	known byte	10	$\simeq 2^{40}$	$\simeq 2^{40}$
Meet-in-the-Middle	3.3	unknown byte	10	$\simeq 2^{60}$	$\simeq 2^{60}$
Meet-in-the-Middle	3.4	fixed unknown byte	5	$\simeq 2^{40}$	$\simeq 2^{40}$
Impossible	4.2	random unknown byte	1000	$\simeq 2^{40}$	$\simeq 2^{40}$
Impossible	4.3	fixed unknown byte	45	$\simeq 2^{40}$	$\simeq 2^{40}$

Organization of the Paper. In Section 2, we recall the backgrounds on AES and on the Piret-Quisquater attack. Then, we describe our meet-in-the-middle and our impossible differential attack on the 7^{th} round in Sections 3 and 4 for AES-128. Finally, in Section 5, we extend these results to the other versions of AES.

2 Backgrounds and Previous Attacks

In this section, we recall the AES operations and we briefly explain how the Piret-Quisquater attack works.

2.1 Description of the AES

AES [15] has a 128-bit input block and can be used with three different key-sizes 128, 192 or 256-bit. It iterates 10 rounds (resp. 12 and 14) for the 128-bit version (resp. for the 192-bit version and for the 256-bit version). According to

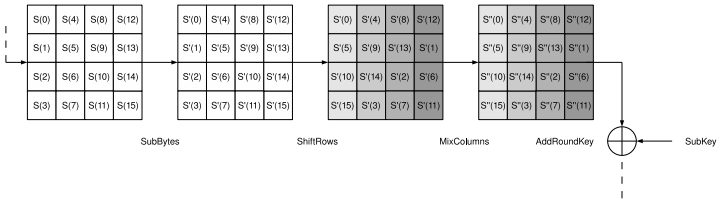

Fig. 1. SubBytes, ShiftRows and MixColumns operations [15]

the bitlength version, we define n as the number of rounds. In Figure 1, we describe one round of the AES which is a composition of the SubBytes, ShiftRows, MixColumns and AddRoundKey operations.

SubBytes (SB). This operation substitutes a value to another one according to the permutation table S-Box, which associates 256 input toward 256 output values. Its goal is to mix non-linearly the bits into one byte.

ShiftRows (SR). This operation changes byte order in the state depending on the row. Each row has its own permutation. The first row changes nothing, the second row is rotated by one position to the left, the third row is rotated by two positions to the left, the fourth row is rotated by three positions to the left.

MixColumns (MC). This operation linearly mixes state bytes by columns and consists in the multiplication of each columns of the state by an MDS matrix (Maximum Distance Separable) in the finite field $GF(2^8)$. We will use the property that, when the input column has one non-null difference in one byte, all the bytes after this operation have a non-null difference.

AddRoundKey Operation (ARK). This operation is only a XOR between intermediate state and the subkey generated by the key schedule.

KeySchedule. The key schedule, which derives the symmetric key K, is composed of two operations, RotWord and SubWord. RotWord is a circular permutation of four elements of one column. SubWord operation corresponds to SubBytes. It is well-known that one subkey of AES-128 allows to retrieve master key K and two consecutive subkeys of AES-192 and AES-256 allow to recover the whole key K. We denote by K_{10} the last subkey of AES-128 and by $K_{10}(0)$ the first byte of the last subkey of AES-128.

2.2 Previous Differential Fault Analysis

In [27], Piret and Quisquater assume a fault injection on one byte during the state computation between the 2 last MixColumns on AES-128 as it is represented in the Figure 2. This attack allows to recover the last subkey in 2^{40} in time and 2^{32} in memory. The idea of the attack consists of expressing 4 differential equation systems at the beginning of the last round state S_{12}. One system is described for each column like equation system (1), where X denotes a non-null

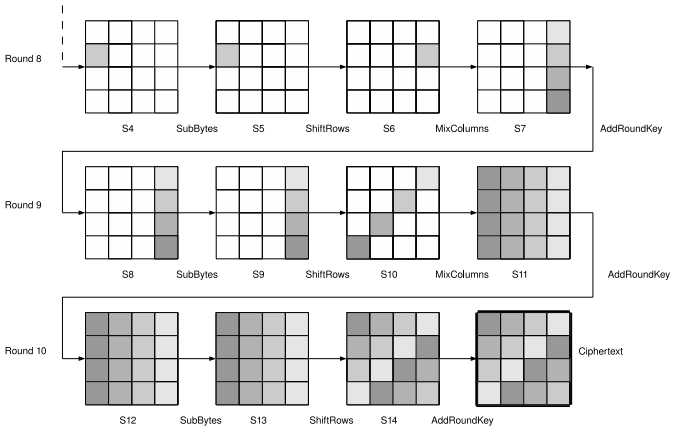

Fig. 2. State-of-the-art differential fault analysis on AES-128

byte difference in state S_{10}. After collecting two couples of correct and faulty ciphertexts, they entirely retrieve the subkey K_{10}.

$$\begin{cases} SB^{-1}(C(0) \oplus K_{10}(0)) \oplus SB^{-1}(\tilde{C}(0) \oplus K_{10}(0)) = X \\ SB^{-1}(C(13) \oplus K_{10}(13)) \oplus SB^{-1}(\tilde{C}(13) \oplus K_{10}(13)) = X \\ SB^{-1}(C(10) \oplus K_{10}(10)) \oplus SB^{-1}(\tilde{C}(10) \oplus K_{10}(10)) = 3X \\ SB^{-1}(C(7) \oplus K_{10}(7)) \oplus SB^{-1}(\tilde{C}(7) \oplus K_{10}(7)) = 2X \end{cases} \quad (1)$$

The right-hand side of the equation system is described one round earlier in annexe B. With only one couple of right and wrong associated results, these equations (1) allow to reduce the possible subkeys from $(2^8)^4 = 2^{32}$ to 2^8 for each equation system. Indeed, according to system (1), there are $(2^8)^4 = 2^{32}$ possible quadruplets of the whole K_{10}:$\{K_{10}(0), K_{10}(13), K_{10}(10), K_{10}(7)\}$. Moreover, there are 2^{40} candidates for $\{X, K_{10}(0), K_{10}(13), K_{10}(10), K_{10}(7)\}$, and the 4 equations give a 32-bit constraint, and consequently, the number of solution is $\frac{2^{40}}{2^{32}} = 2^8$. Then, instead of using another pair of faulty and correct ciphertext as it is done in [27], an exhaustive search can be performed at the end. In the following sections, we will present our differential fault analysis.

3 Meet-in-the-Middle Fault Analysis on AES-128

In our attack, we realize a fault injection on one byte between MixColumns at the 6^{th} round and MixColumns at the 7^{th} round on AES-128. The fault is totally diffused at the whole 10^{th} round as the Figure 3 shows it. This fault analysis requires 10 pairs of correct and faulty ciphertexts. If the attacker knows exactly which byte is faulted, the complexity of the attack is around 2^{40} in time and memory. The overall attack consists in expressing the fault path from

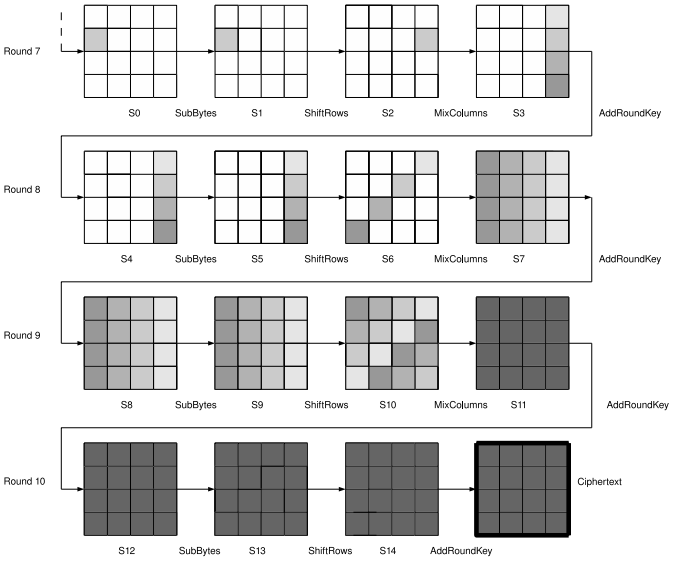

Fig. 3. Overall meet-in-the-middle fault attack on AES-128

the ciphertext to the beginning of the 9^{th} round in the backward direction, and in the forward direction from the fault injection to the beginning of the 9^{th} round. Figure 3 illustrates the error propagation. A classical cryptographic attack against AES, such as the square attack [13], allows to add two rounds after the distinguisher by guessing 5 key bytes. However, this allows to recover one byte of the state. Here, we need to know two bytes of the state, which depend each on 5 different key bytes. By using a clever meet-in-the-middle attack as in the attack of Gilbert and Minier in [16], we are able to recover the key using only 2^{40} space and time. In the following of this section, we explain our differential fault system, our method to retrieve all bytes of the last subkey of AES-128 and its complexity.

3.1 From Fault Path to Differential Fault Equations

The left-hand side of the equation (2) describes the fault path from the ciphertext C at the 10^{th} round toward the state S_8 at the beginning of the 9^{th} round. We obtain:

$$S_8 = SB^{-1}\left(SR^{-1}\left(MC^{-1}\left(ARK^{-1}\left(SB^{-1}\left(SR^{-1}\left(ARK^{-1}(C)\right)\right)\right)\right)\right)\right) \quad (2)$$

We consider each equation byte by byte. The notation $S_8(x)$ denotes the value of the byte x at the state 8. We get the following relations (3) and (4) with $S_8(0)$ and the similar one with $\tilde{S}_8(0)$ as a function of faulty ciphertext $\tilde{C}$, where $MC|_0$ denotes the projection onto the state into the first byte 0.

$$S_8(0) = SB^{-1}\left(MC^{-1}|_0\left(SB^{-1}\left(C(0,7,10,13) \oplus K_{10}(0,7,10,13)\right) \oplus K_9(0,1,2,3)\right)\right) \tag{3}$$

If we define $U_9(0)$ for $\left(MC^{-1}|_0\left(K_9(0,1,2,3)\right)\right)$. Consequently, the byte $S_8(0)$ has the simple expression that depends on 5 unknown bytes, which come key bytes:

$$S_8(0) = SB^{-1}\left(MC^{-1}|_0\left(SB^{-1}\left(C(0,7,10,13) \oplus K_{10}(0,7,10,13)\right)\right) \oplus U_9(0)\right) \tag{4}$$

We obtain a differential equation from the difference between the correct and the faulty state at the end of MixColumns of the 8^{th} round, for example, the first differential equation system (5).

$$\begin{cases} S_8(0) \oplus \tilde{S}_8(0) = X \\ S_8(1) \oplus \tilde{S}_8(1) = X \\ S_8(2) \oplus \tilde{S}_8(2) = 3X \\ S_8(3) \oplus \tilde{S}_8(3) = 2X \end{cases} \tag{5}$$

where X denotes for example the unknown difference of the first column in state S_6 (see Appendix B). We notice that the difference at the end of MixColumns of the 8^{th} round is equal to the difference at the end of AddRoundKey of the 8^{th} round for AES-128. As we mentioned before each equation depends on 5 unknown bytes. We can eliminate the unknown X by considering the following equation:

$$S_8(0) \oplus \tilde{S}_8(0) = S_8(1) \oplus \tilde{S}_8(1). \tag{6}$$

In the next subsection, we will explain how we solve this equation that depend on 80 key bits in time and memory 2^{40} using 10 pairs of faulty and correct ciphertexts.

3.2 Recovery K_{10}

We are interesting in solving 4 difference equations like (5). To simplify the exposition, we will assume that the fault is injected at a known position. Furthermore, the adversary has 10 pairs of correct and faulty ciphertexts and all faults are introduced between the MixColumns at the 6^{th} round and the MixColumns at the 7^{th} round. The constant $U_9(0)$ is invariant for $S_8(x)$ or $\tilde{S}_8(x)$ where $x \in \{0,1,2,3\}$ whatever the plaintext value is.

One idea to solve the system is the following. We consider equation (6) for the ten pairs. Then, we can compute the left hand side for the 2^{40} possible key bytes and store the ten bytes $S_8(0) \oplus \tilde{S}_8(0)$ and the key bytes in a first list. Then, we do the same with the right hand side and store the ten bytes $S_8(1) \oplus \tilde{S}_8(1)$ and the key bytes in a second list. We can merge the two lists, sort them and find collision for the ten bytes. If there is a collision between the two lists, the values of the key bytes gives a solution for the 80 key bits. This simple technique allows

to recover the key bytes in time 2^{45} and memory 2^{41}. We can reduce the space complexity by storing and sorting the list and for each value computed for the second list, we look at if it is also in the first list.

In the following, we present a technique that avoids to increase the time complexity too much by using a hash table.

1. The differential state $S_8(0) \oplus \tilde{S}_8(0)$ is calculated for the 5 pairs of faulty and correct ciphertexts and the results are stored in one hash table according to the values $S_8^i(0) \oplus \tilde{S}_8^i(0)_{1 \leq i \leq 5}$in the one hand, and for the 5 others in the other hand for all possible values of $\{K_{10}(0), K_{10}(7), K_{10}(10), K_{10}(13), U_9(0)\}$. These two hash tables have for input index 5 values of $S_8(0) \oplus \tilde{S}_8(0)$ and for output $\{K_{10}(0), K_{10}(7), K_{10}(10), K_{10}(13),$
$U_9(0)\}$.
2. Then we calculate $\alpha(S_8(1) \oplus \tilde{S}_8(1))$ for all 10 couples of correct and faulty ciphertexts, for all possible hypotheses of $K_{10}(3)$, $K_{10}(6)$, $K_{10}(9)$, $K_{10}(12)$ and $U_9(13)$. Where $U_9(13) = MC^{-1}|_{13}(K_9(12, 13, 14, 15))$ and α is known because fault position is known, i.e $\alpha = 1$. Therefore, we have a relation (7) between $S_8(1)$ and $S_8(0)$ such as:

$$S_8(0) \oplus \tilde{S}_8(0) = \alpha(S_8(1) \oplus \tilde{S}_8(1)) \tag{7}$$

   For each guess for $\{K_{10}(3), K_{10}(6), K_{10}(9), K_{10}(12), U_9(13)\}$, due to the 5 first $S_8(0) \oplus \tilde{S}_8(0)$ indexes and the 5 first $\alpha(S_8(1) \oplus \tilde{S}_8(1))$ calculations, we retrieve a very few potential number of solutions $\{K_{10}(0), K_{10}(7), K_{10}(10),$ $K_{10}(13), U_9(0)\}$ closed to 1 for the first hash table. For the second table, we obtain similar results too. For each table, we make and arrange a linked list for the results of $\{K_{10}(0), K_{10}(7), K_{10}(10), K_{10}(13), U_9(0)\}$. Due to these two arrangements and only for the right values of $\{K_{10}(0), K_{10}(7), K_{10}(10),$ $K_{10}(13), U_9(0)\}$, we have only one intersection between the two linked lists; that is why we only retrieve 8 bytes of K_{10} and the value of α is confirmed for each couple of correct and faulty ciphertexts.
3. We similarly compute $\beta(S_8(2) \oplus \tilde{S}_8(2))$ for the 10 couples of correct and faulty ciphertexts, and for all potential subkey bytes of $K_{10}(2)$, $K_{10}(5)$, $K_{10}(8)$, $K_{10}(15)$ and $U_9(10)$, where $U_9(10) = MC^{-1}|_{10}(K_9(8, 9, 10, 11))$. As step 3, β is known for known fault position, i.e $\beta = \frac{1}{3}$. We obtain the equation (8):

$$S_8(0) \oplus \tilde{S}_8(0) = \beta(S_8(2) \oplus \tilde{S}_8(2)) \tag{8}$$

   Due to previous step, we have knowledge of the value $S_8(0) \oplus \tilde{S}_8(0)$ for the 10 pairs of cipher results. We reuse the previous method of two arranged linked lists. We retrieve $K_{10}(2)$, $K_{10}(5)$, $K_{10}(8)$, $K_{10}(15)$ and $U_9(10)$.
4. As $S_8(2)$, we compute $S_8(3) \oplus \tilde{S}_8(3)$ for the 10 correct and faulty ciphertexts for all possible subkey bytes of $\{K_{10}(1), K_{10}(4), K_{10}(11), K_{10}(14), U_9(7)\}$. Where $U_9(7) = MC^{-1}|_7(K_9(4, 5, 6, 7))$ and $\gamma = \frac{1}{2}$. We have the equation (9):

$$S_8(0) \oplus \tilde{S}_8(0) = \gamma(S_8(3) \oplus \tilde{S}_8(3)) \tag{9}$$

   We also retrieve $K_{10}(1)$, $K_{10}(4)$, $K_{10}(11)$, $K_{10}(14)$, $U_9(7)$ as step 3.

3.3 Cost and Complexity

By the birthday paradox, we have two hash tables with 2^{40} values inside. The complexity of all the system is also 2^{80}. However each equation gives 8-bit constraints, so with ten equations we obtain 80-bit constraints. Consequently, with ten ciphertexts, there is only one solution in our system. Our meet-in-the-middle fault attack requires around 2^{40} in complexity for AES-128: 2^{40} in memory and 3×2^{40} in instructions.

Random Byte Fault Model. In equation (7), α takes on the values $\{\frac{1}{3}, 1, \frac{3}{2}, 2\}$ in case of unknown fault position. Several cases could be studied. In the first one, we know exactly for each faulty ciphertext byte faulty position, we have knowledge of α for each equation. In the second one, we use the same method to inject fault at the same time, we suppose that the same byte is faulted. For consequences, it multiplies by four the computations. In the third case, the worst, we make no assumptions on the location of the fault for each pair of correct and incorrect ciphertexts. In fact for each couple of correct and incorrect results, we need to compute 4 intermediate results. This operation costs 4^{10} values more, it costs too much, i.e 2^{60}.

3.4 Reduction of Memory Requirement

We suppose that an adversary has a *sixtuplet* of the correct message and five faulty ciphertexts, with all five faults on the same byte. In this case, the tool from [9] allows us to find a similar attack but it requires much less memory, 2^{24} instead of 2^{40}.
The previous attack can be schematized as follows :

- Build the four lists, the index 0 corresponds to the correct ciphertext :
  - $L_0 = \left\{\left(K_{10}(0), K_{10}(7), K_{10}(10), K_{10}(13), S_9^0(0)\right)\right\}$
  - $L_1 = \left\{\left(K_{10}(3), K_{10}(6), K_{10}(9), K_{10}(12), S_9^0(1)\right)\right\}$
  - $L_2 = \left\{\left(K_{10}(2), K_{10}(5), K_{10}(8), K_{10}(15), S_9^0(2)\right)\right\}$
  - $L_3 = \left\{\left(K_{10}(1), K_{10}(4), K_{10}(11), K_{10}(14), S_9^0(3)\right)\right\}$
- Each element of L_i allows to deduce unique values for $\Delta S_8^j(i)$, $j = 1, \ldots, 5$ is the index of j^{th} faulty ciphertext.
- Look for collisions since the vector $\left(\Delta S_8^j(0), \ldots, \Delta S_8^j(3)\right)$ must be collinear with a column vector of the matrix of the MixColumn operation.

To reduce memory, we note that we can build each list in beginning with guessing $\Delta S_8^1(0)$ and $\Delta S_8^2(0)$. This operation allows us to partially build the lists and thus save memory.
Building, for example, the list L_0 by assuming that these values are known :

- Build the list $L_0' = \left\{\left(K_{10}(0), S_8^0(0)\right)\right\}$
- Each element of L_0' allows to deduce unique values for :
  - $\Delta S_{10}^j(0)$, $j = 1, 2$
  - $\Delta S_{11}^j(0)$, $j = 1, \ldots, 5$

- Guess $K_{10}(7), K_{10}(10), K_{10}(13)$
  - Deduce $\Delta S_{11}^{j}(1,2,3)$, $j = 1, \ldots, 5$
  - Look in L'_0 corresponding values for $K_{10}(0)$ and $S_8^0(0)$ using $\Delta S_{11}^{j} = MC\left(\Delta S_{10}^{j}\right)$
  - Deduce $\Delta S_8^j(0)$, $j = 3, 4, 5$

L_1, L_2 and L_3 can be built in the same way.
This improvement makes the attack much more feasible. The implementation providing by the tool takes a little bit more than 13 days on a Core 2 Duo E8500 and 900MB of ram to test all possibilities but it can be improved by parallelizing the C code.

4 Impossible Differential Fault Attack on AES-128

In this section, we present a more efficient attack since we do not assume where the fault is provoked and the time complexity is reduced to 2^{41}. However, this fault attack needs more faulty ciphertexts, less than 1000 or 45 depending on the fault model. Our attack is based on the fact that it is impossible to have a zero-difference in state S_{10} in the 9^{th} round just before MixColumns operation; as Phan and Yen mentioned this fact in [26] and developed with an example of the fault injected on the subkey K_7 in the key schedule. This fact is illustrated by the Figure 4. In this section, two principles are associated, the first one impossible differential, which is first published in [21,22], and the second one fault analysis, like [2,26]. Our impossible differential fault analysis corresponds to 5-round impossible differential cryptanalysis attack, which is described in [3]. We firstly present the differential inequation systems, then the retrieval algorithm and in the last part the comparison between the experimental, simulation and theoretical results.

4.1 From Impossible Differential to Inequation System

Due to a well-known property of the differential through the MixColumn operation, all differences between bytes are *not null* at the internal state S_{10} in (10).

$$S_{10}(C) \oplus S_{10}(\tilde{C}) \neq 0 \tag{10}$$

Moreover, we have the following equation (11):

$$S_{10}(C) = MC^{-1}\left(SB^{-1}\left(SR^{-1}(C \oplus K_{10})\right) \oplus K_9\right) \tag{11}$$

We obtain similar equation for $S_{10}(\tilde{C})$. Like the attack below, we have the same simplification with the subkey K_9. The differential equations have the following form (12):

$$S_{10}(C) \oplus S_{10}(\tilde{C}) = MC^{-1}\left(SB^{-1}\left(SR^{-1}(C \oplus K_{10})\right)\right) \oplus MC^{-1}\left(SB^{-1}\left(SR^{-1}(\tilde{C} \oplus K_{10})\right)\right) \tag{12}$$

We execute the same kind of computations as in the previous attack. We analyze column per column. We guess 4 key bytes of K_{10}. Due to the 4 inequalities, we

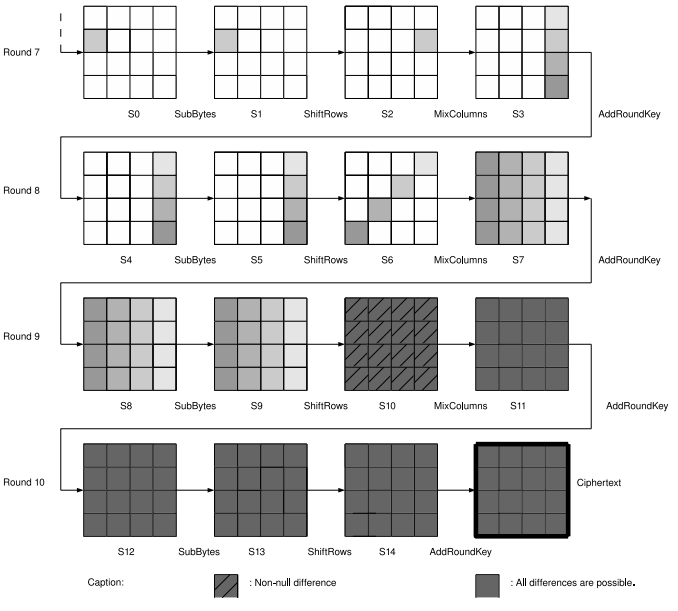

Fig. 4. Overall impossible differential fault attack on AES-128

can filter bad key byte candidates in a list of possible keys. Using many pairs of correct and faulty ciphertexts, we can reduce the possible key space. We reuse four times the no difference computation algorithm for each column of S_{10}. In this attack, the attacker does not use fault position to retrieve the last subkey bytes. The algorithm allows to recover all bytes of the subkey K_{10}. In the case of AES-128, it is enough to retrieve the secret key K.

4.2 Recovery Steps

1. For each pair of correct and incorrect results, we take four guesses for $\{K_{10}(0), K_{10}(13), K_{10}(10), K_{10}(7)\}$. Then we eliminate at each level the key quadruplets which do not satisfy the system (13). We test at each loop all not dismissed quadruplets among 2^{32} possible quadruplets at the beginning.

$$\begin{cases} MC^{-1}|_0(SB^{-1}(C(0) \oplus K_{10}(0))) \oplus MC^{-1}|_0(SB^{-1}(\tilde{C}(0) \oplus K_{10}(0))) \neq 0 \\ MC^{-1}|_1(SB^{-1}(C(13) \oplus K_{10}(13))) \oplus MC^{-1}|_1(SB^{-1}(\tilde{C}(13) \oplus K_{10}(13))) \neq 0 \\ MC^{-1}|_2(SB^{-1}(C(10) \oplus K_{10}(10))) \oplus MC^{-1}|_2(SB^{-1}(\tilde{C}(10) \oplus K_{10}(10))) \neq 0 \\ MC^{-1}|_3(SB^{-1}(C(7) \oplus K_{10}(7))) \oplus MC^{-1}|_3(SB^{-1}(\tilde{C}(7) \oplus K_{10}(7))) \neq 0 \end{cases} \tag{13}$$

2. We repeat previous steps and we retrieve the right quadruplets of K_{10} for each following column.

3. This research could be complemented by an exhaustive search, if only less than 2^{10} possible quadruplets for each column are left. Hence, a global complexity is 2^{40} research operations.

4.3 Property of Recombination

An interesting property of reusing incorrect ciphertexts is described here. The same plaintext is encrypted while fault injection targeted on the same byte. Only MixColumns operation generates collision in one byte, whereas the others do not. Furthermore, if two different inputs of MixColumns only vary on one byte, the two outputs of MixColumns do not collide. For instance, if two different random byte faults ϵ_1 and ϵ_2 are injected on state S_0.

$$\exists! y \in [0,15], \forall \epsilon_1 \neq \epsilon_2, S_0(y) = x \oplus \epsilon_1 \quad S_0(y) = x \oplus \epsilon_2 \tag{14}$$

$\tilde{C}^{(1)}$ is the faulty ciphertext obtained where fault ϵ_1 is injected, similarly, $\tilde{C}^{(2)}$ the faulty ciphertext links to fault ϵ_2. We have the two following facts :

$$S_{10}(C) \oplus S_{10}(\tilde{C}^{(1)}) \neq 0 \tag{15}$$

$$S_{10}(C) \oplus S_{10}(\tilde{C}^{(2)}) \neq 0 \tag{16}$$

Due to equation (14) and the properties of the MixColumns described below, we obtain the following inequation:

$$S_{10}(\tilde{C}^{(1)}) \oplus S_{10}(\tilde{C}^{(2)}) \neq 0 \tag{17}$$

On our test platform, we collect with one correct ciphertext, 5 or 6 different faulty ciphertexts whose faulty bytes are the same.

4.4 Theoretical and Simulation Results

Theoretical Cost and Complexity. The impossible differential algorithm requires 2^{32} guesses as there are 4 unknown key bytes on each column. The probability that all 4 inequations are satisfied equals $(\frac{255}{256})^4$. With one pair of correct and faulty ciphertexts, we eliminate around 2^{26} subkeys of K_{10} amongst 2^{32} possible values of K_{10} for each column: $E = 2^{32} \times (1 - (\frac{255}{256})^4) \simeq 2^{26}$. Each couple could bring the same information about the key than another couple. The recombination of faulty results introduces collision too. Same quadruplets of key bytes are eliminated several times. Two couples of correct and incorrect ciphertexts create an overlap of $\frac{E^2}{2^{32}} \simeq \frac{(2^{26})^2}{2^{32}} = \frac{2^{52}}{2^{32}} = 2^{20}$. We define U_n as the number of rejected quadruplets with n pairs of correct and faulty ciphertexts with the following recursive formula, where $U_0 = 0$:

$$U_{n+1} = 2^{26} + U_n(1 - 2^{-6}). \tag{18}$$

In solving recurrence in previous equation 18, we obtain the following equation:

$$U_n = 2^{32} - 2^{32}(1 - 2^{-6})^n. \tag{19}$$

The recovery algorithm of the impossible difference stops where $U_n \geq 2^{32} - 2^{10}$. That is why, due to equation 19,

$$n \geq -22\log(2)/\log(1 - 2^{-6}) \Leftrightarrow n \geq 968. \tag{20}$$

Simulation Results. We obtain around 2^{26} eliminated quadruplets of bytes for each pair. We also retrieve the calculated overlap of 2^{20} between two pairs. Considering the random byte fault model, we need on average around 1000 couples of correct and faulty ciphertexts with performing an exhaustive search on 2^{40} possible subkeys at the end. In the case of recombination based on the fixed byte fault model, due to collision results, our attack only requires about 45 faulty ciphertexts with the same plaintext among the 256 possible ciphertexts: $\binom{45+1}{2} = \frac{45\times 46}{2} = 1035 > 1000$. It is also possible to combine classical resolution with several recombinations.

5 Extension to AES-192 and AES-256

Introducing fault between the MixColumns of the 6^{th} round and the MixColumns of the 7^{th} round on AES-128 amounts to injecting fault between the MixColumns of the 8^{th} round and MixColumns of the 9^{th} round on AES-192, and between the 10^{th} round and the 11^{th} round on AES-256. Because faults are injected one round before all previous papers, we have access at the same time at subkeys K_n and K_{n-1} with the same differential path.

5.1 Meet-in-the-Middle Fault Analysis on AES-192 and AES-256

We extend the previous concepts for AES-192 and AES-256 without more faulty ciphertexts than AES-128. We use the meet-in-the-middle algorithm in order to recover: $\{K_n(4), K_n(1), K_n(14), K_n(11), U_{n-1}(7)\}$, $\{K_n(8), K_n(5), K_n(2), K_n(15), U_{n-1}(10)\}$, $\{K_n(0), K_n(7), K_n(10), K_n(13), U_{n-1}(0)\}$ and $\{K_n(3), K_n(6), K_n(9), K_n(12), U_{n-1}(13)\}$. We obtain 2 tables which contain $S_8(0) \oplus \tilde{S}_8(0)$ for 5 couples of correct and incorrect results. We compute $S_8(1) \oplus \tilde{S}_8(1)$, $S_8(2) \oplus \tilde{S}_8(2)$ and $S_8(3) \oplus \tilde{S}_8(3)$. By hypothesis, we know fault position for each faulty ciphertext, it means that α, β and γ are known for all equations. Due to these computations, we retrieve all bytes of the subkey K_n. We write the differential equations $S_8(5)$, $S_8(10)$ and $S_8(15)$ as a function of the same 4 bytes of K_{n-1}. Then we also write system of $S_8(6)$, $S_8(11)$ and $S_8(12)$ as a function of the same 4 bytes of K_{n-1}, $S_8(7)$, $S_8(8)$ and $S_8(13)$ as a function of 4 bytes of K_{n-1} and $S_8(4)$, $S_8(9)$ and $S_8(14)$ as a function of 4 bytes of K_{n-1}. We inject the 16 computed bytes of K_n in the previous equations like (5). We recognize the form of

Piret and Quisquater equations in our ones (21), that is why we apply Piret and Quisquater resolution in our recovery method.

$$\begin{cases} SB^{-1}\left(A \oplus U_{n-1}(4)\right) \oplus SB^{-1}(\tilde{A} \oplus U_{n-1}(4)) = Y \\ SB^{-1}\left(B \oplus U_{n-1}(1)\right) \oplus SB^{-1}(\tilde{B} \oplus U_{n-1}(1)) = 3Y \\ SB^{-1}\left(C \oplus U_{n-1}(14)\right) \oplus SB^{-1}(\tilde{C} \oplus U_{n-1}(14)) = 2Y \\ SB^{-1}\left(D \oplus U_{n-1}(11)\right) \oplus SB^{-1}(\tilde{D} \oplus U_{n-1}(11)) = Y \end{cases} \tag{21}$$

The values $\{A, B, C, D\}$ are known values at this stage and only depend on the correct ciphertext and K_n. The values $\{\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}\}$ are known values too and only depend on the faulty ciphertext and K_n. Using 3 generalizations of Piret and Quisquater equation systems allow to recover the subkey U_{n-1}, because we have already retrieved $\{U_{n-1}(0), U_{n-1}(13), U_{n-1}(10), U_{n-1}(7)\}$. Then we resolve 4 systems of 4 equations in using the Gauss' method. Each equation describes MixColumns inverse operation with unknown outputs, in order to recover all bytes of K_{n-1}. This scenario costs around 2^{40} in complexity for AES-192 or AES-256 divided in 2^{40} for memory and 3×2^{40} for operation code like AES-128, plus 2^{40} for Piret and Quisquater resolution.

5.2 Impossible Differential Fault Analysis on AES-192 and AES-256

In the cases of AES-192 and AES-256, we do not need more fault than AES-128 if no exhaustive search is realized. However, we have to collect couples until all bytes of the subkey K_n are retrieved. We reuse the equation systems (5) of the first attack, because both attacks consider fault injection between the same MixColumns. Now, we obtain as the previous subsection the systems (21), thanks to which we know all bytes of K_n. In order to retrieve all bytes of the subkey K_{n-1}, we use 4 Piret and Quisquater generalization. This fault attack is achieved with a complexity around 2^{42}, because Piret and Quisquater generalization has the same cost as Piret and Quisquater attack [27] described in the second part of this paper.

6 Conclusion

We have presented two different attacks on the $n-3^{th}$ round of AES as it is shown in Table 1. The first attack implies random fault byte on known or fixed position for AES-128, AES-192 or AES-256. The second attack involves random fault byte too with less complexity for AES-128. The first one costs around 2^{42} and requires 10 pairs of correct and faulty ciphertexts, its improvement 5 pairs and costs 2^{40} whereas the second one around 2^{40} deals with 1000 couples. Moreover, we can associate the first analysis to solve the second subpart of the second analysis. In this case, a differential fault analysis could be performed on AES-128, AES-192 and AES-256 with a random fault injected between the $n-4^{th}$ and the $n-3^{th}$ MixColumns. Current state-of-the-art countermeasure

consists on protecting the three first rounds and the three last rounds of AES. All operations inside round need to be protected and state between rounds too. In order to defeat our fault analysis, all AES-128 rounds need to be protected against fault attacks. Considering AES-192 and AES-256, at least the last 5 rounds and the first 5 rounds need to be protected against fault analysis.

Acknowledgments. We would like to thank Nicolas Guillermin and the anonymous reviewers for their helpful and valuable comments and discussions.

References

1. Anderson, R.J., Kuhn, M.G.: Low Cost Attacks on Tamper Resistant Devices. In: Christianson, B., Lomas, M. (eds.) Security Protocols 1997. LNCS, vol. 1361, pp. 125–136. Springer, Heidelberg (1998)
2. Biham, E., Granboulan, L., Nguyen, P.Q.: Impossible fault analysis of RC4 and differential fault analysis of RC4. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 359–367. Springer, Heidelberg (2005)
3. Biham, E., Keller, N.: Cryptanalysis of Reduced Variants of Rijndael. In: 3rd AES Conference, New York, USA (2000)
4. 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)
5. Biryukov, A., Khovratovich, D.: Two New Techniques of Side-Channel Cryptanalysis. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 195–208. Springer, Heidelberg (2007)
6. Bloemer, 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)
7. Bogdanov, A.: Improved Side-Channel Collision Attacks on AES. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 84–95. Springer, Heidelberg (2007)
8. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
9. Bouillaguet, C., Derbez, P., Fouque, P.-A.: Automatic Search of Attacks on Round-Reduced AES and Applications. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 169–187. Springer, Heidelberg (2011)
10. 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, pp. 118–129. Springer, Heidelberg (2003)
11. Choukri, H., Tunstall, M.: Round Reduction Using Faults. In: Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2005, pp. 13–24 (2005)
12. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M.: Passive and Active Combined Attacks on AES Combining Fault Attacks and Side Channel Analysis. In: FDTC, pp. 10–19 (2010)

13. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Springer, Heidelberg (2002)
14. Dusart, P., Letourneux, G., Vivolo, O.: Differential Fault Analysis on A.E.S. In: Zhou, J., Yung, M., Han, Y. (eds.) ACNS 2003. LNCS, vol. 2846, pp. 293–306. Springer, Heidelberg (2003)
15. FIPS. Advanced Encryption Standard (AES). pub-NIST (November 2001)
16. Gilbert, H., Minier, M.: A Collision Attack on 7 Rounds of Rijndael. In: AES Candidate Conference. LNCS, pp. 230–241. Springer, Heidelberg (2000)
17. Giraud, C.: DFA on AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2005. LNCS, vol. 3373, pp. 27–41. Springer, Heidelberg (2005)
18. Hamid, H.B.-E., Choukri, H., Tunstall, D.N.M., Whelan, C.: The Sorcerer's Apprentice Guide to Fault Attacks (2004), http://eprint.iacr.org/2004/100.pdf
19. Kermani, M.M., Reyhani-Masoleh, A.: A Lightweight Concurrent Fault Detection Scheme for the AES S-Boxes Using Normal Basis. In: Oswald and Rohatgi [25], pp. 113–129
20. Kim, C.H.: Differential Fault Analysis against AES-192 and AES-256 with Minimal Faults. In: Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 3–9 (2010)
21. Knudsen, L.R.: DEAL - a 128 bit block cipher. In: Technical report 151, Departement of Informatics, University of Bergen, Norway (1998)
22. Knudsen, L.R.: DEAL - a 128 bit block cipher. In: AES Round 1 Technical Evaluation, NIST (1998)
23. 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)
24. Mukhopadhyay, D.: An Improved Fault Based Attack of the Advanced Encryption Standard. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 421–434. Springer, Heidelberg (2009)
25. Oswald, E., Rohatgi, P. (eds.): CHES 2008. LNCS, vol. 5154. Springer, Heidelberg (2008)
26. Phan, R.C.-W., Yen, S.-M.: Amplifying Side-Channel Attacks with Techniques from Block Cipher Cryptanalysis. In: Domingo-Ferrer, J., Posegga, J., Schreckling, D. (eds.) CARDIS 2006. LNCS, vol. 3928, pp. 135–150. Springer, Heidelberg (2006)
27. Piret, G., Quisquater, J.-J.: A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 77–88. Springer, Heidelberg (2003)
28. Rivain, M.: Differential Fault Analysis on DES Middle Rounds. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 457–469. Springer, Heidelberg (2009)
29. Satoh, A., Sugawara, T., Homma, N., Aoki, T.: High-Performance Concurrent Error Detection Scheme for AES Hardware. In: Oswald and Rohatgi [25], pp. 100–112
30. Schramm, K., Leander, G., Felke, P., Paar, C.: A Collision-Attack on AES: Combining Side Channel- and Differential-Attack. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 163–175. Springer, Heidelberg (2004)
31. Takahashi, J., Fukunaga, T., Yamakoshi, K.: DFA Mechanism on the AES Key Schedule. In: FDTC 2007: Proceedings of the Workshop on Fault Diagnosis and Tolerance in Cryptography, pp. 62–74. IEEE Computer Society, Los Alamitos (2007)
32. Tunstall, M., Mukhopadhyay, D.: Differential Fault Analysis of the Advanced Encryption Standard using a Single Fault. Cryptology ePrint Archive, Report 2009/575 (2009), http://eprint.iacr.org/

A Difference Path from the 10^{th} to the 9^{th} Round for AES-128

Due to fault path from the ciphertext to the beginning of the 9^{th} round, we give the following relations between bytes at different steps for the Meet-in-the-Middle attack. We obtain the following system of 4 equations, where $U_9(a, b, c, d) = MC^{-1}(K_9(a, b, c, d))$, for AES-128 from the 10^{th} to the 9^{th} round, for AES-192 from the 12^{th} to the 11^{th} and for AES-256 from the 14^{th} to the 13^{th}:

$$S_8(0,5,10,15) = SB^{-1}\left(MC^{-1}\left(SB^{-1}\left(C(0,7,10,13) \oplus K_{10}(0,7,10,13)\right)\right) \oplus U_9(0,1,2,3)\right) \quad (22)$$

$$S_8(1,6,11,12) = SB^{-1}\left(MC^{-1}\left(SB^{-1}\left(C(3,6,9,12) \oplus K_{10}(3,6,9,12)\right)\right) \oplus U_9(12,13,14,15)\right) \quad (23)$$

$$S_8(2,7,8,13) = SB^{-1}\left(MC^{-1}\left(SB^{-1}\left(C(2,5,8,15) \oplus K_{10}(2,5,8,15)\right)\right) \oplus U_9(8,9,10,11)\right) \quad (24)$$

$$S_8(3,4,9,14) = SB^{-1}\left(MC^{-1}\left(SB^{-1}\left(C(1,4,11,14) \oplus K_{10}(1,4,11,14)\right)\right) \oplus U_9(4,5,6,7)\right) \quad (25)$$

B Difference Path from the 7^{th} towards the 8^{th} Round on AES-128

Fault on one byte among bytes $\{0, 5, 10, 15\}$ at the 7^{th} round on AES-128 produces case 1, fault on one byte among $\{3, 4, 9, 14\}$ produces case 2, fault on one byte among $\{2, 7, 8, 13\}$ produces case 3 and fault on one byte among $\{1, 6, 11, 12\}$ produces case 4. All different cases are presented in Figure 5. We obtain same behavior with fault injected at the 9^{th} round of AES-192 and at the 11^{th} round of AES-256.

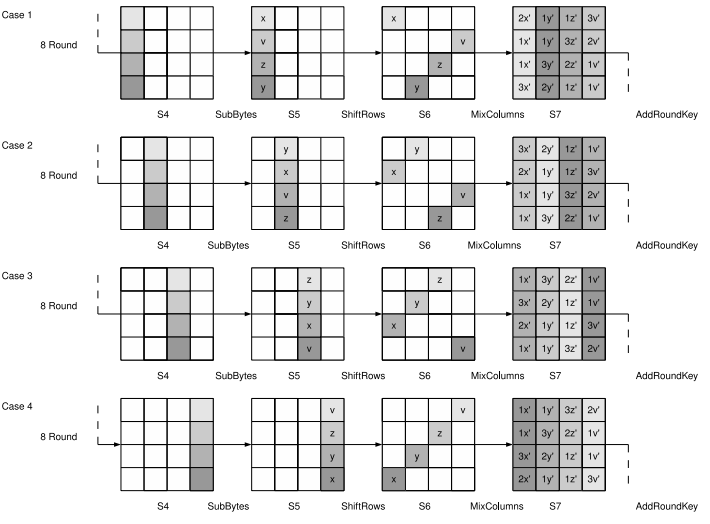

Fig. 5. Difference path during the 8^{th} round for the four different AES-128 cases