Combined Attacks on the AES Key Schedule

François Dassance and Alexandre Venelli

Inside Secure
Avenue Victoire, 13790 Rousset, France
{fdassance,avenelli}@insidefr.com

Abstract. We present new combined attacks on the AES key schedule based on the work of Roche *et al.* [16]. The main drawbacks of the original attack are: the need for high repeatability of the fault, a very particular fault model and a very high complexity of the key recovery algorithm. We consider more practical fault models, we obtain improved key recovery algorithms and we present more attack paths for combined attacks on AES. We propose to inject faults on the different operations of the key schedule instead of the key state of round 9 or the corresponding data state. We also consider fault injections in AES constants such as the RCon or the affine transformation of the SubWord. By corrupting these constants, the attacker can easily deduce the value of the error. The key recovery complexity can then be greatly improved. Notably, we can obtain a complexity identical to a classical differential side-channel attack. Our attacks defeat most AES implementations secure against both high-order side-channel attacks and fault attacks.

Keywords: Side-channel analysis, Fault analysis, Combined attack, AES

1 Introduction

Side-channel attacks are a major threat to most cryptosystems implemented on embedded devices. The main principle of these attacks is to exploit a physical leakage from the device in order to find some information on the used secret. Side-channel attacks are categorized in *Simple Side-Channel Analysis* (SSCA) and *Differential Side-Channel Analysis* (DSCA). Another kind of physical attack is the *Fault Analysis* (FA) that evaluates the faulty behavior of a cryptosystem to learn information on the secret. Since the first paper of Kocher *et al.* [8] on DSCA, many countermeasures have been proposed to thwart physical attacks on different cryptosystems.

Recently, the principle of combined attacks has been proposed in order to attack a cryptosystem protected against both side-channel analysis and fault analysis. Amiel *et al.* [1] first clearly presented this new kind of attack on an RSA implementation secure against SSCA and FA. Combined attacks on the *Advanced Encryption Standard* (AES) [10] were also developed in [15, 3, 16].

The most common countermeasure to protect block cipher implementations, like AES, against DSCA is masking techniques where random values are used to hide intermediate variables. Simultaneously, protections against FA were also developed for block ciphers. The most intuitive countermeasure often consists in the duplication of the block cipher or the computation of its inverse operation. However combined attacks offer a new attack path. They exploit information, through either SSCA or DSCA, obtained from a fault before most of FA countermeasures in the literature are able to detect them. Hence, new protections and new implementation methods have to be considered.

In this article, we present new combined attacks on AES and more particularly on the AES key schedule. Based on the work of Roche *et al.* [16], we evaluate different attack scenarios and implementations of the key schedule that lead to powerful and efficient combined attacks. Moreover, some of our attack paths require additional countermeasures compared to Roche *et al.*'s work.

The paper is organized as follows. In Section 2, we give a brief description of the AES. Section 3 summarizes the main combined attacks proposed in the literature on asymmetric and symmetric cryptosystems. We also recall Roche *et al.*'s attack and give remarks on its efficiency considering a somewhat more realistic fault model. In Section 4, we describe our proposed combined attacks on the AES key schedule. We target different internal operations and we consider different fault models. We report countermeasures against these attacks on AES in Section 5. We conclude this article in Section 6.

2 The AES Algorithm

The AES is defined for 128-bit blocks and key sizes 128, 192 and 256 bits. The 128-bit plaintext is viewed as a 4×4 byte matrix, called state, bytes corresponding in some way to elements of $\mathbb{F}_{2^8}$.

The AES operates on states by iterating transformation rounds. The initial round consists in the $\mathsf{AddRoundKey}$ operation, the next rounds consist in applying successively the transformations $\mathsf{SubBytes}$, $\mathsf{ShiftRows}$, $\mathsf{MixColumns}$ and $\mathsf{AddRoundKey}$, but the last round omits the $\mathsf{MixColumns}$ transformation. The $\mathsf{SubBytes}$ is the main building block of AES regarding the side-channel aspect. Each byte of the state matrix is replaced by its substitute in an SBox. This SBox is the composition of two transformations: an inversion in $\mathbb{F}_{2^8}$ and an affine transformation. $\mathsf{ShiftRows}$ is a cyclic shift operation on each of the four rows of the state. The first row is unchanged, the second is cyclically shifted by one byte to the left, the third by two bytes and the fourth by three bytes. $\mathsf{MixColumns}$ considers each column of the state matrix as coefficients of a degree three polynomial and multiplies them modulo $z^4 + 1$ with a fixed polynomial. $\mathsf{AddRoundKey}$ is a bit-wise XOR operation between the state and the round key.

The round keys are derived from the original key with the AES key scheduling algorithm. The key schedule has a recursive structure that uses bytes of the previous column of the key state, bytes of the column of a previous round key and round constants RCon_r with r the round number. A non-linear part is realized using a $\mathsf{SubBytes}$ operation on a column, this operation is also called $\mathsf{SubWord}$. An additional cyclic rotation, noted $\mathsf{RotWord}$, is performed within the column. Figure 1 details the last three rounds of the key schedule algorithm for AES-128.

In the rest of this paper, we focus our work on AES-128. For more details on AES, one can refer to the AES standard [10].

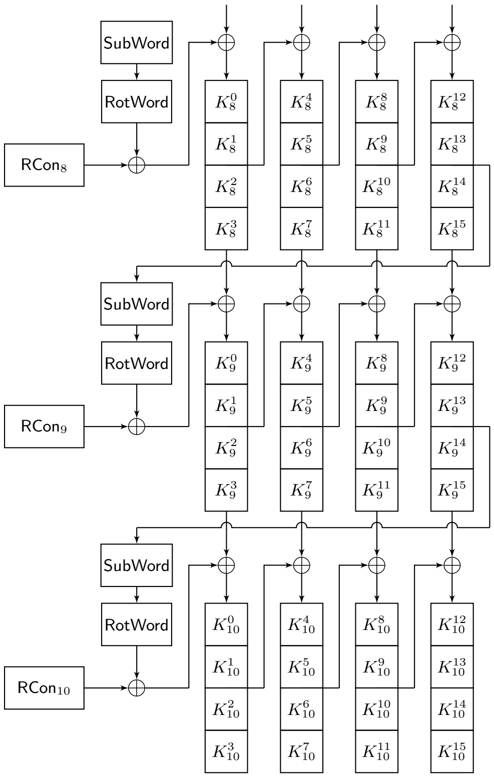

Fig. 1. Last three rounds of the AES-128 key schedule.

3 Related Work on Combined Attacks

Recently, one of the main area of research in side-channel analysis is combined attacks. A combined attack exploits leakage information from both a Fault Analysis (FA) and a classical side-channel attack like *Simple Side-Channel Analysis* (SSCA) or *Differential Side-Channel Analysis* (DSCA). Both symmetric and asymmetric cryptosystems have been shown vulnerable to this attack. We briefly review the combined attacks proposed in the literature.

3.1 Combined Attacks on Asymmetric Cryptosystems

Amiel *et al.* [1] combine a fault attack with an SSCA in order to break a modular exponentiation that is supposedly secure against faults and SSCA. The authors attack a left-to-right atomic square-and-multiply algorithm that is SSCA-resistant. By injecting a fault in one of the registers at the beginning of the exponentiation, they detect information on the secret exponent by SSCA. Hence, the FA protection that is present at the end of the algorithm cannot prevent the SSCA leakage that has already occurred during the computation. In [1], the authors propose a countermeasure called *Detect and Derive* based on the principle of infective computation. However, it was shown vulnerable in [17]. In this paper, Schmidt *et al.* propose an exponentiation algorithm, as well as a scalar multiplication algorithm, also based on infective computation. The idea is to be able to detect a fault as soon as it happens and corrupt the data if necessary so that no leakage information is available. The principle of the attack of Amiel *et al.* seems to be applicable to any classic left-to-right atomic algorithm, either exponentiation or scalar multiplication. Right-to-left implementations as well as regular algorithms seem resistant against this kind of combined attack.

In [5], Fan *et al.* study the case of combined attacks specially targeting elliptic curve scalar multiplication. Using the properties of elliptic curves, they develop a powerful attack that can defeat atomic and regular algorithms. In order to perform the attack, one needs to choose a particular input point of the scalar multiplication. By injecting a fault after the initial point verification, the attacker is then able to obtain a point with a small order. During the scalar multiplication, computations with the faulted point will end up on the infinity point which is particularly visible by SSCA in most implementations. The attacker is then able to find information on the secret scalar.

A recent DSCA countermeasure proposed by Dupaquis and Venelli in [4] seems to be an efficient generic countermeasure against combined attacks. Using randomized reduction algorithms, one can refresh masks within a modular exponentiation or scalar multiplication for a small overhead. Hence, an attacker is not able to find by SSCA a particular pattern induced by a fault. However, as the randomization added by this countermeasure is based on the form of the modulus, it is not sufficient to prevent combined attacks on a scalar multiplication for sparse modulus, *e.g.* like NIST elliptic curves [9].

3.2 Combined Attacks on Symmetric Cryptosystems

The combined attacks proposed on symmetric cryptosystems all combine fault attacks with DSCA. In [15], Robisson and Manet introduce the *Differential Behavioral Analysis* (DBA) which combines a safe-error attack and DSCA. We recall that safe-error attacks are based on the information whether the device has a normal behavior or not in presence of a fault. The authors show that the DBA can break an AES hardware implementation, however it is ineffective on masked implementations.

Clavier *et al.* [3] successfully attack an AES protected with first-order masking. More generally, their combined attack reduces the DSCA countermeasure of one order. This attack is focused on the first round and requires the knowledge of faulty ciphertexts. Hence, a fault countermeasure like the inverse computation or the duplication of the AES prevents this attack.

Recently, Roche *et al.* [16] propose the first combined attack on AES implementation that can defeat both a boolean masking of any order and a fault countermeasure. The attack targets the last round, more precisely, it targets the key state of round 9. If the attacker is able to inject a fault with good enough repeatability, he has to perform statistical tests using three simultaneous hypothesis to recover one byte of the key. Depending on the fault model considered, either *bit-flip* or *stuck-at*, the fault repeatability can be impacted, hence the number of power consumption curves required to perform the DSCA. Considering the *bit-flip* model, an attacker can defeat a boolean masking of any order without any overhead in the attack. However, if one considers the *stuck-at* model, masking induces a repeatability automatically divided by 2. As our attacks are based on the work of Roche *et al.*, we recall the principle of their attack and we propose improvements depending on the fault model considered.

3.3 Remarks on Roche *et al.*'s Combined Attack

Algorithm 1 Roche *et al.*'s combined attack key retrieval algorithm.

Input: N triplets of valid-faulty ciphertexts and the side-channel information of the faulted execution: $(C_1, \tilde{C}_1, \mathcal{L}_1), \ldots, (C_N, \tilde{C}_N, \mathcal{L}_N)$
Output: The guessed round 10 key k_{10}

for $j = 0$ to 15 **do**
  for $\left(k_{10}^j, e_9^j, e_{10}^j\right) = (0,0,0)$ to $(255,255,255)$ **do**
    for $i = 1$ to N **do**
      $V_i = \phi\left(\mathsf{SubBytes}\left(\mathsf{SubBytes}^{-1}\left(C_i^j \oplus k_{10}^j\right) \oplus e_9^j\right) \oplus k_{10}^j \oplus e_{10}^j\right)$
    end for
    $\sigma_{\left(k_{10}^j, e_9^j, e_{10}^j\right)} = \mathcal{A}\left(\left(\mathcal{L}_i\right)_{1 \le i \le N}, \left(V_i\right)_{1 \le i \le N}\right)$
  end for
  The maximal $\sigma_{\left(k_{10}^j, e_9^j, e_{10}^j\right)}$ for $e_9^j \neq 0$ corresponds to the guessed value k_{10}^j
end for
return k_{10}

We detail the attack of Roche *et al.* in Algorithm 1. We denote by ϕ the considered leakage model function, *e.g.* Hamming weight or identity. Let $\mathcal{A}$ be any DSCA statistical function, *e.g.* Pearson correlation [2] or mutual information [6]. Algorithm 1 has a complexity of $\underline{2^{28}\mathcal{A}}$ on N leakage measurements. We recall the main relation of Roche *et al.*'s attack:

$$\tilde{C}_i^j = \mathsf{SubBytes}\left(\mathsf{SubBytes}^{-1}\left(C_i^j \oplus k_{10}^j\right) \oplus e_9^j\right) \oplus k_{10}^j \oplus e_{10}^j \ , \qquad (1)$$

with $\tilde{C}_i^j$ the faulty ciphertext byte j of message i, C_i^j the byte j of the correct ciphertext of message i, k_{10}^j the guessed byte j of the round key K_{10}, e_9^j the error in the byte j of K_9 and e_{10}^j the error in byte j of K_{10}. We denote by K_r^j the byte j of the correct round key r. In order to retrieve the 16 bytes of K_{10} with Algorithm 1, we need to assume that for each triplet $(C_i, \tilde{C}_i, \mathcal{L}_i)$ the attacker was able to inject a fault modifying the 16 bytes of K_9, where $\mathcal{L}_i$ is the side-channel measurement of the faulty message i.

On the efficiency of the combined attack. We fix that N side-channel leakages are sufficient to obtain a distinguishable correct triplet $(k_{10}^j, e_9^j, e_{10}^j)$ for any byte j for a given fault repeatability using the test $\mathcal{A}$. The efficiency of the combined attack depends on the complexity of its key retrieval algorithm as well as the value of N that impacts the complexity of $\mathcal{A}$. We recall that the complexity of a classical DSCA is $2^{12}\mathcal{A}$ to retrieve 16 bytes compared to $2^{28}\mathcal{A}$ of Algorithm 1. In order to break a masked implementation, an higher-order DSCA has the same key retrieval complexity of $2^{12}\mathcal{A}$ but the high number of required plaintext N can dramatically reduce the efficiency of the attack. On the other hand, this combined attack is able to break a masked implementation of any order with a simple DSCA targeting first-order leakages, hence requiring much less plaintexts. As an example, we can note that in the experimentations of [16, Section 5.3], a combined attack with 90% success rate can be obtained with $N < 2000$ plaintexts considering a fault repeatability only slightly above 50% and a standard deviation of noise $\sigma = 5$. In [12, Section 5] a second-order DSCA is successful on a first order boolean masking with 45 000 plaintexts on a similar setup (considering a 90% success rate and $\sigma \approx 5$).

Note also that this combined attack defeats implementations protected against faults using a single-fault injection mechanism, *i.e.* no complex multi-fault setup is required to bypass the protection [18].

On the fault injection assumptions. In [16], the authors consider a fault injection mechanism that faults the 16 bytes of K_9 at each execution of AES (whatever the pattern of the fault within each byte). The attacker only needs N fault injections hence N side-channel leakages measurements to perform Algorithm 1. The complexity of the key retrieval algorithm is $2^{28}\mathcal{A}$ with N faults injected.

Injecting a fault on the 16 bytes of K_9 during one execution of AES is not trivial on most implementations. For example, the bytes of the key state K_9 may be spread over a large area on the chip and the spot size of the laser may be so large that the device would almost certainly cease to function as a result of the disruption to surrounding circuitry. Note also that if the 16 bytes of the key state are not computed in one cycle, a fault injection using a clock glitch cannot impact all bytes.

The single byte fault model. In the following, we consider a more realistic scenario where the attacker can inject a random fault within one known byte of index j (a single byte fault model). The attacker has to perform 16 fault injection campaigns each time retrieving N side-channel leakages. Then, he can perform Algorithm 1 on each of the bytes with each of the 16 sets of side-channel leakages. However the attacker has to perform $16N$ faults as well as side-channel measurements. Note that if the injected fault modifies m bytes of the key (a m-byte fault model), the attacker needs $\lceil 16/m \rceil N$ faults and side-channel measurements.

Remark 1. Considering the single byte fault model, if a fault is injected in one of the bytes K_9^j, then we have that $e_{10}^j = e_9^j$ for $0 \leq j \leq 15$. This property can reduce the complexity of Algorithm 1 to $\underline{2^{20}\mathcal{A}}$ to find the complete key. If the fault is wider than a single byte, simple relations between e_{10}^j and e_9^j can be found as long as the fault only impacts the bytes $(K_9^j)_{0\leq j\leq 11}$. We can also obtain a complexity of $2^{16}\mathcal{A}$ for each of these 12 bytes. If a fault affects multiple bytes with one byte of the last column of K_9, *i.e.* $(K_9^j)_{12\leq j\leq 15}$, the $\mathsf{SubWord}$ transformation on this column makes it impossible to find relationships between errors. For example, consider that K_9^0 and K_9^{13} are simultaneously faulted. From the equations of the key schedule for K_{10}, we note that the error on K_9^{13} goes in the input of the $\mathsf{SubWord}$ and is then XOR-ed to K_{10}^0. Hence, one cannot find easy relations between the values of errors in round 9 and 10.

In this fault model, the main drawbacks of this combined attack are:

- the need for a high repeatability of fault effects,
- a great number of injected faults,
- a key recovery algorithm of very high complexity.

In the following, we propose new attacks that improve these points.

4 Combined Attacks on the AES Key Schedule

The structure of the AES key schedule can be used in order to decrease the number of faults. First, we target operations of the key schedule. Then, we consider a fault injection into an AES constant that is XOR-ed with multiple bytes in the key schedule. Hence, the recursive structure of the key schedule assures that a single fault will affect several bytes. As we attack known constants, the attacker can also deduce the value of his fault injection, or at least a small subset of possible values. Generally, AES constants are not masked using any DSCA countermeasure, hence whatever the fault model, the fault repeatability is not affected. Finally, we consider the case of permanent faults that obviously help with the repeatability issue.

4.1 Targeting the Recursive Structure of the Key Schedule Round

We recall the key schedule equations to compute the bytes of the round key K_9:

$$K_9^0 = K_8^0 \oplus \mathsf{RCon}_9 \oplus \mathsf{SubBytes}\left(K_8^{13}\right) \qquad K_9^1 = K_8^1 \oplus \mathsf{SubBytes}\left(K_8^{14}\right)$$
$$K_9^2 = K_8^2 \oplus \mathsf{SubBytes}\left(K_8^{15}\right) \qquad K_9^3 = K_8^3 \oplus \mathsf{SubBytes}\left(K_8^{12}\right)$$
$$K_9^j = K_8^j \oplus K_9^{j-4} \quad \text{for } 4 \le j \le 15 \ .$$

Due to the recursive property of the key schedule, a fault injected during the computation of K_9^0 is propagated in K_9^4, K_9^8 and K_9^{12}. Hence, we obtain the following relations for given errors $e_9^0, e_9^1, e_9^2, e_9^3$ during the computation of the first 4 bytes of K_9:

$$e_9^0 = e_9^j \quad \text{for } j \in \{4, 8, 12\} \qquad e_9^1 = e_9^j \quad \text{for } j \in \{5, 9, 13\}$$
$$e_9^2 = e_9^j \quad \text{for } j \in \{6, 10, 14\} \qquad e_9^3 = e_9^j \quad \text{for } j \in \{7, 11, 15\} \ .$$

We can also deduce relations on the errors e_{10}^j on K_{10}. Indeed, if K_9^0 is faulted during its computation we have from the key schedule equations of round 10:

$$e_{10}^j = e_9^j \quad \text{for } j \in \{0, 8\} \qquad e_{10}^j = 0 \quad \text{for } j \in \{4, 12\} \ .$$

Similar relations can easily be obtained if a fault is injected during the computation of K_9^1, K_9^2 or K_9^3.

The attacker only needs $4N$ faults and side-channel measurements to recover the complete key. The key recovery algorithm can also be improved from $2^{20}\mathcal{A}$ of the original attack (see Remark 1). In order to retrieve the byte K_{10}^0, the attacker needs a loop on k_{10}^0 and e_9^0, as the error e_{10}^0 is known from the error on round 9. Hence, the complexity to find this byte is $2^{16}\mathcal{A}$. Once e_9^0 is found, the attacker directly determines the errors for the bytes K_9^4, K_9^8 and K_9^{12}. Thus, a simple loop on k_{10}^j with $j = 4, 8, 12$ for each of those bytes is necessary. Then, the complexity for these 3 bytes is only $2^8\mathcal{A}$. The same method applies to the bytes K_9^1, K_9^2 and K_9^3. The complexity for the complete key is $(4 \times (2^{16} + 3 \times 2^8))\mathcal{A} = \underline{(2^{18} + 3 \times 2^{10})\mathcal{A}}$.

4.2 Targeting the RCon

Single byte random fault model. First, we assume a fault that has a random effect on a byte. We recall the equations to compute the first bytes of the columns of K_9:

$$K_9^0 = K_8^0 \oplus \mathsf{RCon}_9 \oplus \mathsf{SubWord}\left(K_8^{13}\right) \qquad K_9^4 = K_8^4 \oplus K_9^0$$
$$K_9^8 = K_8^8 \oplus K_9^4 \qquad K_9^{12} = K_8^{12} \oplus K_9^8 \ .$$

Note that RCon_9 affects 4 bytes of K_9, more precisely RCon_9 is implicitly XOR-ed with the bytes K_9^0, K_9^4, K_9^8, K_9^{12}. If we consider that the attacker can inject a fault noted ϵ_9 into the round constant of round 9 of the key schedule, then this fault will affect the 4 bytes of K_9 in the exact same way. Figure 2 represents the impact of such a fault on the end of the key schedule. An attack scenario similar to the one presented previously in § 4.1 can be applied to recover the bytes K_{10}^0, K_{10}^4, K_{10}^8 and K_{10}^{12}. The attacker only needs N faults and side-channel measurements. The complexity of the key recovery is only $\underline{(2^{16} + 3 \times 2^8)\mathcal{A}}$ to retrieve 4 bytes. Finally, note that the fault can be injected at any time before the computation of the round 9 of the key schedule. The fault can also have a permanent effect on RCon_9. In this case, the attacker only needs one fault injection and N side-channel measurements to perform the attack.

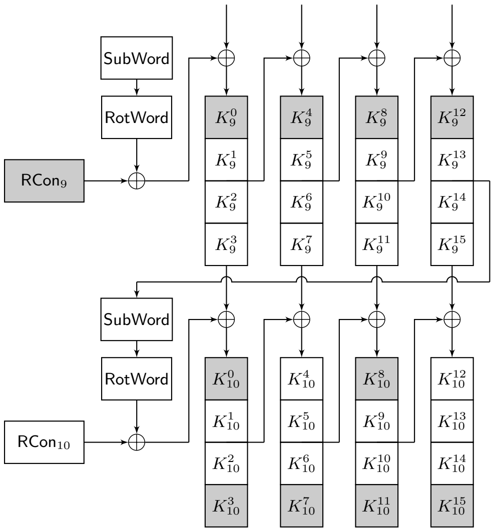

Fig. 2. Effect of a fault injection on RCon_9 on the end of the key schedule. The color gray represents faulty elements.

Single byte known fault model. An adversary can use the fact that he attacks a known constant in order to deduce the value of the error he injects. He is then able to lower the complexity of the key recovery algorithm. Suppose that the attacker has characterized his fault injection setup but is not able to know which bit he targets, he can greatly reduce the number of hypothesis on the error on K_9. For example, in a single bit *stuck-at* 0 or 1 model, the error ϵ_9 can take only 4 possible values due to the value of $\mathsf{RCon}_9 = \texttt{0x1B}$. In a single bit *bit-flip* model, the error can take 8 different values. Finally, if the attacker knows which bit he targets, he can directly deduce the value of the error. In this last case, the attacker knows the values of $e_9^0 = e_9^4 = e_9^8 = e_9^{12} = \epsilon_9$, hence he knows the values of the corresponding errors on K_{10} as shown in § 4.1. The complexity of the key recovery algorithm is approximately $\underline{2^{10}\mathcal{A}}$ to recover 4 bytes which is the cost of a classical first-order DSCA on 4 bytes. A permanent fault on RCon_9 has the same consequences on the number of faults and side-channel measurements as previously noted.

4.3 Targeting the Affine Transformation

In this section, we consider implementations of AES where the SubWord (and SubBytes) operations are computed as an inverse in $\mathbb{F}_{2^8}$ followed by the affine transformation. Most first-order DSCA countermeasures use this type of implementation, *e.g.* [11, 19]. More importantly, most higher-order DSCA countermeasures compute the inverse and the affine transformation separately, *e.g.* [14, 13, 7]. In this case, we can find a similar but more efficient approach than previously.

Let $\mathsf{Inv}_{\mathbb{F}_{2^8}}$ be the inverse in $\mathbb{F}_{2^8}$ with the convention $\mathsf{Inv}_{\mathbb{F}_{2^8}}(0) = 0$. Let f be the affine transformation of the SubBytes such as: $f(x) = \omega(x) + \delta$ defined in the standard basis by $\Omega.X + \Delta$ as:

$$\Omega = \begin{bmatrix} 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1\\ 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1 \end{bmatrix} \quad \text{and} \quad \Delta = \begin{bmatrix} 0\,1\,1\,0\,0\,0\,1\,1 \end{bmatrix}.$$

Practically in the AES, an element $X \in \mathbb{F}_{2^8}$ is identified to the integer $X_7 2^7 + X_6 2^6 + \cdots + X_0$. Hence, Δ is considered as $\Delta = \mathtt{0x63}$ in a real implementation. The SubBytes operation then becomes: $\mathsf{SubBytes}(X) = \Omega.\mathsf{Inv}_{\mathbb{F}_{2^8}}(X) \oplus \Delta$. As the SubWord operation of the key schedule corresponds to the SubBytes on 4 bytes, the same remark also applies.

We write again the first key schedule equations for K_9 detailing the SubBytes:

$$\begin{aligned} K_9^0 &= K_8^0 \oplus \mathsf{RCon}_9 \oplus \Omega.\mathsf{Inv}_{\mathbb{F}_{2^8}}\left(K_8^{13}\right) \oplus \Delta \\ K_9^1 &= K_8^1 \oplus \Omega.\mathsf{Inv}_{\mathbb{F}_{2^8}}\left(K_8^{14}\right) \oplus \Delta \\ K_9^2 &= K_8^2 \oplus \Omega.\mathsf{Inv}_{\mathbb{F}_{2^8}}\left(K_8^{15}\right) \oplus \Delta \\ K_9^3 &= K_8^3 \oplus \Omega.\mathsf{Inv}_{\mathbb{F}_{2^8}}\left(K_8^{12}\right) \oplus \Delta \ . \end{aligned}$$

Single byte random fault model. Different attack scenarios are possible depending on the implementation and on the duration of the effect of the fault.

In the first scenario, we consider the effect of the fault transient. An attack similar to the one presented in § 4.1 can be applied by targeting Δ. The complexities are identical. To retrieve the complete key, the attacker needs $4N$ faults and side-channel measurements. The complexity of the key recovery algorithm is $\underline{(2^{18} + 3 \times 2^{10})\mathcal{A}}$.

In a second scenario, we consider that the implementation has a variable Δ_{SW} for the SubWord different from the Δ_{SB} of the SubBytes. In the case of an hardware implementation, we can similarly consider that the SubBytes and SubWord logical structures are located at different physical positions on the chip. Suppose that an attacker can inject a fault in Δ_{SW} that lasts until the end of AES. Then, this error affects both K_9 and K_{10}. We can find relations between the errors in round 9 and 10 to simplify the key recovery algorithm. First, let $\widetilde{\Delta_{SW}} = \Delta_{SW} \oplus e_{SW}$ where e_{SW} is the error injected in Δ_{SW}. From the round 9 key schedule equations, we have:

$$\widetilde{K}_9^j = K_9^j \oplus e_{SW} \quad \text{for } 0 \leq j \leq 15 \ ,$$

with $\widetilde{K}_r^j$ the faulty key byte j of round key r. From the fact that we have the same error e_{SW} in the SubWord of round 10, we have:

$$\begin{aligned}
\widetilde{K}_{10}^0 &= \widetilde{K}_9^0 \oplus \mathsf{RCon}_{10} \oplus \mathsf{SubBytes}\left(\widetilde{K}_9^{13}\right) \oplus e_{SW} \\
&= K_{10}^0 \oplus \left(\mathsf{SubBytes}\left(K_9^{13} \oplus e_{SW}\right) \oplus \mathsf{SubBytes}\left(K_9^{13}\right)\right) \\
&= K_{10}^0 \oplus e_{10}^0 \\
\widetilde{K}_{10}^1 &= K_{10}^1 \oplus \left(\mathsf{SubBytes}\left(K_9^{14} \oplus e_{SW}\right) \oplus \mathsf{SubBytes}\left(K_9^{14}\right)\right) \\
&= K_{10}^1 \oplus e_{10}^1 \\
\widetilde{K}_{10}^2 &= K_{10}^2 \oplus \left(\mathsf{SubBytes}\left(K_9^{15} \oplus e_{SW}\right) \oplus \mathsf{SubBytes}\left(K_9^{15}\right)\right) \\
&= K_{10}^2 \oplus e_{10}^2 \\
\widetilde{K}_{10}^3 &= K_{10}^3 \oplus \left(\mathsf{SubBytes}\left(K_9^{12} \oplus e_{SW}\right) \oplus \mathsf{SubBytes}\left(K_9^{12}\right)\right) \\
&= K_{10}^3 \oplus e_{10}^3 \ .
\end{aligned}$$

Hence we obtain the following errors for the other bytes of K_{10}, for $j \in \{0, 1, 2, 3\}$:

$$\begin{aligned}
e_{10}^{j+4} &= e_{10}^{j+12} = e_{10}^j \oplus e_{SW} \\
e_{10}^{j+8} &= e_{10}^j \ .
\end{aligned}$$

With only N faults and side-channel measurements, the attacker can retrieve all 16 bytes of K_{10}. Using the previous relations, we can also simplify Algorithm 1. A loop on the three hypothesis $(k_{10}^0, e_9^0, e_{10}^0)$ is necessary as there is no known relation between e_9^0 and e_{10}^0, *i.e.* a complexity of $2^{24}\mathcal{A}$. Once e_9^0 is found, the attacker knows that $e_9^0 = e_9^j$ for $1 \leq j \leq 15$. However e_{10}^1, e_{10}^2 and e_{10}^3 are still unknown. The complexity to retrieve the corresponding bytes is $3 \times 2^{16}\mathcal{A}$. Finally, the attacker knows e_9^j and e_{10}^j for $4 \leq j \leq 15$ from the previous relations. The overall complexity to retrieve all the bytes is $\underline{(2^{24} + 3 \times 2^{16} + 3 \times 2^{10})\mathcal{A}}$.

In the last scenario, we consider that the SubWord and the SubBytes use the same variable Δ, or the same logical structure in an hardware implementation.

If the fault in Δ lasts until the end of AES, it affects K_9, K_{10} and the $\mathsf{SubBytes}$ in the data path of the last round. The same observations as in the last scenario applies. However, as the data path is also modified we need to adapt Equation (1), corresponding to Line 4 of Algorithm 1, into :

$$\mathsf{SubBytes}\left(\mathsf{SubBytes}^{-1}\left(C_i^j \oplus k_{10}^j\right) \oplus e_9^j\right) \oplus e_9^j \oplus k_{10}^j \oplus e_{10}^j \ .$$

The error e_9^j is the same at the output of $\mathsf{SubWord}$ in round 9 of key schedule and in the last $\mathsf{SubBytes}$ of the data path. Hence, we just have to add a XOR with e_9^j after the $\mathsf{SubBytes}$ in order to retrieve the key. The number of faults and the complexity of the key recovery algorithm are identical to the previous scenario.

Single byte known fault model. As in the RCon case, we know the value of the constant Δ we want to fault. If the attacker characterized his fault injection setup, he should be able to reduce to a small subset (or directly determine) the value of the error. In this case, the attacker knows e_9^j for $0 \leq j \leq 15$. Considering the first scenario where the attacker temporarily faults a Δ in one of the $\mathsf{SubBytes}$, he needs to obtain $4N$ faults and side-channel measurements as well as a key recovery complexity of $\underline{2^{12}\mathcal{A}}$ to retrieve the key. We obtain a combined attack with complexity identical to a classical DSCA. In the second and third scenarios, the key recovery complexity becomes $\underline{(2^{20} + 3 \times 2^{10})\mathcal{A}}$ with only N faults and side-channel measurements in order to retrieve the key.

Remark 2. The attack of [16] targets a key state or data state possibly masked with a high-order DSCA countermeasure. Hence, if the fault injection setup has a *stuck-at* effect, the repeatability of the fault is divided by 2. However, most AES implementations do not consider AES constants masked. Thus our attacks on RCon and the affine transformation are not affected by the decrease in fault repeatability of a *stuck-at* model.

We summarize in Table 1 the different attacks proposed in this paper. For each scenario, we note the fault model considered, the number of necessary faults and the number of statistical tests $\mathcal{A}$ performed in the key recovery algorithm.

5 Countermeasures

An intrinsic property of Roche *et al.*'s attack and our propositions is that it requires that an attacker can cipher the same plaintext twice. Note that this is not possible in certain cryptographic protocols, notably in banking transactions. Indeed, a counter can be added to the plaintext and make this kind of combined attack impossible.

As already pointed out in [16, Remark 6], the original combined attack of Roche *et al.* does not necessarily target the key K_9 but can also target the data state in the round 9. In that case, the key K_{10} is correct. Thus one cannot only check the coherence of the key at the end of an AES.

Table 1. Summary of the complexities of the different combined attacks, where $\mathcal{A}$ is a DSCA statistical test used on N measurements to recover a byte of key. We recall that the complexity of a classical DSCA is $2^{12}\mathcal{A}$ to retrieve 16 bytes.

Attack	**# faults**	**# $\mathcal{A}$**
Key state K_9 [16]		
– Transient random fault on 16 bytes	N	2^{28}
Key state K_9 § 3.3		
– Transient random fault on 1 byte	$16N$	2^{20}
Recursive structure of the key schedule § 4.1		
– Transient random fault on 1 byte	$4N$	$2^{18} + 3 \times 2^{10}$
RCon § 4.2[a]		
– Transient random fault on 1 byte	N	$2^{16} + 3 \times 2^{8}$
– Transient known fault on 1 byte	N	2^{10}
– Permanent random fault on 1 byte	1	$2^{16} + 3 \times 2^{8}$
– Permanent known fault on 1 byte	1	2^{10}
$\mathsf{SubWord}$ § 4.3		
– Transient random fault on 1 byte	$4N$	$2^{18} + 3 \times 2^{10}$
– Transient known fault on 1 byte	$4N$	2^{12}
– Permanent random fault on 1 byte	N	$2^{24} + 3 \times 2^{16} + 3 \times 2^{10}$
– Permanent known fault on 1 byte	N	$2^{20} + 3 \times 2^{10}$

[a] Attack on 4 bytes.

The aim of this combined attack is to find correlations on the plain ciphertext. Hence, an efficient countermeasure consists in storing the ciphertext still masked while either an inverse operation or a duplication of the operation is performed [16]. Consider $C_1 \oplus M_1$ and $C_2 \oplus M_2$ the ciphertexts C_1 and C_2 of the same message masked with M_1 and M_2 respectively. The final coherence verification is then performed as $(C_1 \oplus M_1) \oplus M_2 \stackrel{?}{=} (C_2 \oplus M_2) \oplus M_1$. This countermeasure thwarts most our attacks.

Note that if a permanent fault is injected in the affine transformation, a coherence check consisting of two AES encryptions would not detect the error. A check using an AES encryption and a decryption would detect it as the affine transformation differs between the two process. However, even if the error is not detected, this does not offer a valid attack path as the data state would be corrupted from the first round of the AES. Hence, the attacker is not able to find relationships between errors at the rounds 9 and 10.

However, a coherence check cannot detect a permanent fault on RCon_9 as presented in § 4.2. The value RCon_9 is XOR-ed in the key schedule and it is symmetrically used in the encryption and decryption process. Hence, performing two encryptions or an AES and its inverse does not detect a faulty RCon_9 while our combined attack is still feasible. A possible countermeasure would be to perform a known answer test or to compute an integrity check on the value of RCon_9 in a software implementation.

6 Conclusion

We propose in this paper combined attacks on the AES key schedule that defeat classical AES implementations secure against high-order DSCA and fault attacks. We present attacks on the different parts of the key schedule that use its structure in order to reduce the number of faults injected as well as the key recovery complexity. Contrary to the original attack, the performance of our proposed attacks on AES constants is not impacted whether a *stuck-at* or a *bit-flip* model is considered when attacking a masked implementation. Moreover, the key recovery algorithm of [16] can be greatly improved from the original complexity of 2^{28} DSCA statistical tests on N measurements to only 2^{12} which is equivalent to a classical DSCA key retrieval. The coherence check proposed in [16] defeat most of our propositions. However, our attack path on RCon_9 requires a particular integrity check.

References

1. Amiel, F., Villegas, K., Feix, B., Marcel, L.: Passive and Active Combined Attacks: Combining Fault Attacks and Side Channel Analysis. In: Breveglieri, I., Gueron, S., Koren, I., Naccache, D., Seifert, J. (eds.) Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 92–102. IEEE Computer Society, Washington, DC, USA (2007)
2. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004, Lecture Notes in Computer Science, vol. 3156, pp. 135–152. Springer Berlin / Heidelberg (2004)
3. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M.: Passive and Active Combined Attacks on AES: Combining Fault Attacks and Side Channel Analysis. In: Breveglieri, L., Joye, M., Koren, I., Naccache, D., Verbauwhede, I. (eds.) Workshop on Fault Diagnosis and Tolerance in Cryptography. pp. 10–19. IEEE Computer Society, Los Alamitos, CA, USA (2010)
4. Dupaquis, V., Venelli, A.: Redundant Modular Reduction Algorithms. In: Prouff, E. (ed.) Smart Card Research and Advanced Applications, Lecture Notes in Computer Science, vol. 7079, pp. 102–114. Springer Berlin / Heidelberg (2011)
5. Fan, J., Gierlichs, B., Vercauteren, F.: To Infinity and Beyond: Combined Attack on ECC Using Points of Low Order. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011, Lecture Notes in Computer Science, vol. 6917, pp. 143–159. Springer Berlin / Heidelberg (2011)
6. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual Information Analysis. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems CHES 2008, Lecture Notes in Computer Science, vol. 5154, pp. 426–442. Springer Berlin / Heidelberg (2008)
7. Kim, H., Hong, S., Lim, J.: A Fast and Provably Secure Higher-Order Masking of AES S-Box. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011, Lecture Notes in Computer Science, vol. 6917, pp. 95–107. Springer Berlin / Heidelberg (2011)
8. Kocher, P., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) Advances in Cryptology - CRYPTO' 99, Lecture Notes in Computer Science, vol. 1666, pp. 789–789. Springer Berlin / Heidelberg (1999)

9. National Institute Standards and Technology: Digital Signature Standard (DSS). Publication 186–2 (2000)
10. National Institute Standards and Technology: Advanced Encryption Standard (AES). Publication 197 (2001)
11. Oswald, E., Schramm, K.: An Efficient Masking Scheme for AES Software Implementations. In: Song, J.S., Kwon, T., Yung, M. (eds.) Information Security Applications, Lecture Notes in Computer Science, vol. 3786, pp. 292–305. Springer Berlin / Heidelberg (2006)
12. Prouff, E., Roche, T.: Attack on a Higher-Order Masking of the AES Based on Homographic Functions. In: Gong, G., Gupta, K. (eds.) Progress in Cryptology - INDOCRYPT 2010, Lecture Notes in Computer Science, vol. 6498, pp. 262–281. Springer Berlin / Heidelberg (2010)
13. Prouff, E., Roche, T.: Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In: Preneel, B., Takagi, T. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2011, Lecture Notes in Computer Science, vol. 6917, pp. 63–78. Springer Berlin / Heidelberg (2011)
14. Rivain, M., Prouff, E.: Provably Secure Higher-Order Masking of AES. In: Mangard, S., Standaert, F.X. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2010, Lecture Notes in Computer Science, vol. 6225, pp. 413–427. Springer Berlin / Heidelberg (2010)
15. Robisson, B., Manet, P.: Differential Behavioral Analysis. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007, Lecture Notes in Computer Science, vol. 4727, pp. 413–426. Springer Berlin / Heidelberg (2007)
16. Roche, T., Lomn, V., Khalfallah, K.: Combined Fault and Side-Channel Attack on Protected Implementations of AES. In: Prouff, E. (ed.) Smart Card Research and Advanced Applications, Lecture Notes in Computer Science, vol. 7079, pp. 65–83. Springer Berlin / Heidelberg (2011)
17. Schmidt, J.M., Tunstall, M., Avanzi, R., Kizhvatov, I., Kasper, T., Oswald, D.: Combined Implementation Attack Resistant Exponentiation. In: Abdalla, M., Barreto, P. (eds.) Progress in Cryptology - LATINCRYPT 2010, Lecture Notes in Computer Science, vol. 6212, pp. 305–322. Springer Berlin / Heidelberg (2010)
18. Trichina, E., Korkikyan, R.: Multi Fault Laser Attacks on Protected CRT-RSA. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2010 Workshop on. pp. 75–86 (2010)
19. Trichina, E., Korkishko, L.: Secure and Efficient AES Software Implementation for Smart Cards. In: Lim, C., Yung, M. (eds.) Information Security Applications, Lecture Notes in Computer Science, vol. 3325, pp. 425–439. Springer Berlin / Heidelberg (2005)