Fault Attacks on AES with Faulty Ciphertexts Only

Thomas Fuhr, Eliane Jaulmes, Victor Lomné and Adrian Thillard
ANSSI
51, Bd de la Tour-Maubourg, 75700 Paris 07 SP, France
firstname.lastname@ssi.gouv.fr

***Abstract*—Classical Fault Attacks often require the ability to encrypt twice the same plaintext, in order to get one or several pairs of correct and faulty ciphertexts corresponding to the same message. This observation led some designers to think that a randomized mode of operation may be sufficient to protect block cipher encryption against this kind of threat.**

In this paper, we consider the case where the adversary neither chooses nor knows the input messages, and has only access to the faulty ciphertexts. In this context, we are able to describe several attacks against AES-128 by using non uniform fault models. Our attacks target the last 4 rounds and allow to recover the correct key with practical time complexity, using a limited number of faulty ciphertexts. This work highlights the need for dedicated fault attack countermeasures in secure embedded systems.

***Keywords*-Fault Attacks, AES**

I. Introduction

It is today well known that cryptographic algorithms are susceptible to Fault Attacks (FA for short). Indeed, since the seminal work of Boneh *et al.* [7], a lot of papers have proposed FA on several widely used cryptographic primitives, including symmetric ciphers such as the DES [5] or the AES [17] as well as asymmetric operations like RSA [7] or Elliptic Curve Cryptography [4].

FA consist in inducing a logical error through a physical mean in one of the intermediate variables of a cryptographic operation, and to exploit the erroneous result to get information on the key. The means to inject a logical error can consist in over/under-powering the device during a short instant, in tampering its clock, or in injecting a light beam or an electro-magnetic field inside the device [2], [16], [19].

Several cryptanalytic methods have been developed to exploit erroneous results in order to retrieve the key. In the Differential Fault Analysis (DFA) [5], one runs a cryptographic function twice on the same input and introduces a fault towards the end of one of the computations. Then, one can retrieve information on the key from the differences between the correct and the faulty results.

The Safe Error Attack (SEA) [20] consists in sticking part of the cryptographic secret to a known value. Then, the observation of a collision on the result of a correct computation and a faulted one on identical inputs leaks information on the secret.

In Collision Fault Analysis [6], one runs a cryptographic operation on two related inputs, and introduces a fault near the beginning of one of the computations. The adversary then exploits cases where a collision on the outputs occurs.

A common requirement of all these FA is the necessity of processing two inputs that are either identical or related, in order for the adversary to generate pairs of correct/faulty ciphertexts. Therefore, one requires the ability to control the input of a cryptographic operation, which classifies them as chosen-plaintext attacks. Some of these FA require only one pair of correct/faulty outputs obtained from the same input, whereas others require several pairs to retrieve the secret key[1].

Some protocols exploit this requirement as a countermeasure against FA: they are constructed such that it is not possible for the adversary to encrypt/sign twice the same message (or two related messages). An example of such a construction

[1] Some other attacks (like in [5]) do not need the correct ciphertexts. However, they require the knowledge of the plaintexts.

can be achieved by padding a random value to the message. They are thus inherently protected against chosen-plaintext fault attacks. One can cite the protocol described by Guilley *et al.* [13] or EMV signatures [1].

At CHES 2009, Coron *et al.* exhibited a fault attack against RSA signatures with partially unknown messages [9], improved in [10]. Thus they showed that protecting RSA signatures against FA at the protocol level is generally not sufficient.

1) Our results: in this work, we propose new Fault Attacks on AES, succeeding with *random* and *unknown* plaintexts. The adversary only requires a collection of *faulty ciphertexts* encrypted with the same key. Unlike most traditional attacks, we consider models in which the fault injection introduces a *bias* on a target variable (as in [15]), whereas the incoming variable is uniformly distributed due to the cipher properties.

First we propose two FA targeting the end of the 9^{th} and the 8^{th} AES round. These attacks exploit several fault models that include different non uniformly distributed faults. The adversary has to collect several faulty ciphertexts where one byte of the State has been modified. Depending on the bias introduced on the considered State byte, we propose three different distinguishers allowing to recover the secret key.

Then we propose two FA targeting the State of the 7^{th} and the 6^{th} AES round just after the MixColumns. In these two attacks, we are using a more restrained fault model where the adversary is able to stick some bytes to a constant unknown value. The attack targeting the round 7 requires to target one diagonal of the State, whereas the attack on the round 6 needs to perturb three diagonals of the State.

2) The paper is organized as follows: in Section II we first briefly recall the AES block cipher and define the notations we will use. Then we describe the two attacks on rounds 9 and 8 using non uniformly distributed faults in Section III. We restrain the model to faults sticking bytes of the State to constant unknown values in order to attack rounds 7 and 6 in Section IV. Finally conclusions are drawn in the Section V.

II. Background and notations

A. A brief overview of the AES

In this paper we focus on applying our attack strategy to the AES. We give only a partial description of this algorithm, that includes the features of AES that are needed to understand our attacks, see [12] for a complete specification.

The AES is a block cipher with 128-bit blocks. In our attacks we consider the 128-bit key variant. The block is divided into an array of 4×4 bytes, indexed $0, \ldots, 15$. The quadruples of bytes $(4i, 4i+1, 4i+2, 4i+3)$ are called *columns* and the quadruples of bytes $(i, i+4, i+8, i+12)$ are called *rows*. In the following we will also speak of *diagonals* when considering bytes $(i, 4{+}(i{+}1 \bmod 4), 8+(i+2 \bmod 4), 12+(i+3 \bmod 4))$.

An AES round is the composition of the following four operations:

- SubBytes (SB): is the application of a fixed permutation to each of the 16 bytes of the State;
- ShiftRows (SR): is a circular shift on the four rows of the State. More precisely, row i is transformed by a circular shift on bytes by i positions to the left;
- MixColumns (MC): is a linear bijection on the four columns in parallel;
- AddRoundKey (AK): consists in XOR-ing a 128-bit round key to the 128-bit State.

A first key K_0 is XOR-ed to the plaintext, before the first round. In the last round, no MixColumns operation is applied. We do not provide a description of the round key generation algorithm that is used to compute the 11 round keys from the encryption key. We only remind the reader that the encryption key (and thus all the sequence of round keys) can be fully recovered from any of the round keys.

Finally, we remind the reader of a well-known property: let us consider a value of the State S and a round key K_r. The result of the application of a MixColumns followed by the round key addition to S is $MC(S) \oplus K_r$. As MC is linear, we can write:

$$\begin{aligned} MC(S) \oplus K_r &= MC(S) \oplus MC(MC^{-1}(K_r)) \\ &= MC(S \oplus MC^{-1}(K_r)). \end{aligned} \tag{1}$$

This shows that the sequence of operations MixColumns followed by AddRoundKey K_r is equivalent to first the XOR of $MC^{-1}(K_r)$ followed by a MixColumns.

B. Notations

We denote by K the secret key used by the algorithm, and by K_r the key used in the r-th round. Assuming several encryptions, we denote by P^i the i-th plaintext encrypted using K, and by C^i the corresponding ciphertext. Furthermore, we denote the intermediate State obtained after the SubBytes (SB), ShiftRows (SR), MixColumns (MC), AddRoundKey (AK) operation of the round r respectively by Ssb_r^i, Ssr_r^i, Smc_r^i, Sak_r^i. For any State S, we denote by $\tilde{S}$ its faulty counterpart. For any hypothesis $\hat{K}$ made on K, we denote by $\hat{S}$ the hypothetical value of S. Finally, we denote by $S[j]$ the j-th byte of S.

As seen in Section II-A, the AddRoundKey operation can be computed before the MixColumns, provided that the round key K_r is replaced by $MC^{-1}(K_r)$. We denote by $Smod_r^i$ the intermediate value of the State in this case, which corresponds to $Ssr_r^i \oplus MC^{-1}(K_r)$.

III. Attacks on 9-th and 8-th round

A. Fault model and general principle

In this section, it is assumed that the adversary is able to encrypt an unknown collection of plaintexts $\{P^1, P^2, \cdots, P^n\}$, and to inject a fault on a particular byte $Sak_r^i[j]$ for all $i \in [1, n]$. We suppose moreover that he has access to the resulting collection $\{\tilde{C}_1, \tilde{C}_2, \cdots, \tilde{C}_n\}$ of faulty ciphers, but not to any correct ciphertext.

Our considered fault model covers a large set of faults: the injected fault is only assumed to disturb a particular byte of Sak_r^i in such a way that the distribution of the faulty value $\tilde{Sak}_r^i[j]$ is biased, even when $Sak_r^i[j]$ is uniform. We will however distinguish three different cases, illustrating the degree of control of the attacker on the injected fault:

1) *Perfect control.* The attacker perfectly knows the statistical distribution of the faulty value $\tilde{Sak}_r^i[j]$.
2) *Partial control.* The attacker has some partial information on the distribution of the faulty value $\tilde{Sak}_r^i[j]$. In particular, we will consider the case where he knows the logical effect induced by the perturbation. For the sake of clarity, we will focus on the case where this logical effect is a AND or OR function (this means that we have $\tilde{Sak}_r^i[j] = Sak_r^i$ AND e^i or $\tilde{Sak}_r^i[j] = Sak_r^i$ OR e^i, where e is an unknown error byte).
3) *No control.* The attacker has no information about the distribution of the faulty value $\tilde{Sak}_r^i[j]$, except that it is non uniform.

Our strategy to recover the correct key is somewhat similar to the one used in [18]. Several hypotheses $\hat{K}$ are made on secret bytes, in order to retrieve an hypothetical value which distribution is strongly biased. In our particular context however, we do not need any correct ciphertexts to compute the hypothetical value $\hat{Sak}_r^i[j]$. Depending on the degree of control of the attacker on the fault distribution, we use different distinguishers to retrieve the correct key.

I/ *Maximum likelihood.* When the attacker perfectly knows the distribution of the faulty value, he can then distinguish the correct key by using a maximum likelihood approach, that is, by looking for the hypothesis $\hat{K}$ which maximize $l(\hat{K})$, where:

$$l(\hat{K}) = \prod_{i=1}^{n} P(\tilde{Sak}_r^i = \hat{Sak}_r^i).$$

II/ *Minimal (resp. Maximal) mean Hamming weight.* The AND (resp. OR) function inherently introduces a bias in the distribution of each bit of $\tilde{Sak}_i^r[j]$. With no further knowledge, each bit of $\tilde{Sak}_i^r[j]$ equals 0 with a strong (resp. weak) probability. We hence propose to search for the minimal (resp. maximum) mean of Hamming weights (HW), that is, to look for the hypothesis $\hat{K}$ which minimizes (resp. maximizes) $h(\hat{K})$, where:

$$h(\hat{K}) = \frac{1}{n}\sum_{i=1}^{n} HW(\hat{Sak}_r^i).$$

III/ *Square Euclidean Imbalance (SEI).* When the attacker only knows that the distribution is biased, we propose to look for the strongest possible bias. To this end we use the Squared Euclidean Imbalance (like in [18]) to measure the distance between the obtained hypothetical distributions and the uniform distribution. We therefore look for the hypothesis $\hat{K}$ maximizing $s(\hat{K})$, such that:

$$s(\hat{K}) = \sum_{\delta=0}^{255}\left(\frac{\#\{i|\hat{Sak}_r^i[j]=\delta\}}{n} - \frac{1}{256}\right)^2.$$

During the rest of this section, we will use the three following fault models to illustrate the efficiency of the proposed distinguishers:

a) the so-called *stuck at* 0 fault model with probability 1:

$$\tilde{Sak}_i^r \quad = \quad Sak_i^r \texttt{ AND } 0 \text{ with proba. } 1$$

b) the *stuck at* 0 fault model with probability $\frac{1}{2}$ (where e is uniformly distributed in $[0, 255]$):

$$\begin{cases} \tilde{Sak}_i^r \quad = \quad Sak_i^r \texttt{ AND } 0 \text{ with proba. } \frac{1}{2} \\ \tilde{Sak}_i^r \quad = \quad Sak_i^r \texttt{ AND } e \text{ with proba. } \frac{1}{2} \end{cases}$$

c) the *stuck at* model with an unknown and random value e (where e is uniformly distributed in $[0, 255]$):

$$\tilde{Sak}_i^r \quad = \quad Sak_i^r \texttt{ AND } e \text{ with proba. } 1.$$

B. Attack on the 9*-th round*

1) Fault location: The fault is injected just after the penultimate `AddRoundKey` operation of the `AES`, modifying Sak_9.

2) Description of the attack: We can express $\tilde{Sak}_9^i$ as a function of $\tilde{C}^i$ and one byte of K_{10}:

$$\tilde{Sak}_9^i = SB^{-1} \circ SR^{-1}(\tilde{C}^i \oplus K_{10}). \qquad (2)$$

Each byte of $\tilde{Sak}_9^i$ can hence be deduced using only one hypothesis on a particular byte of K_{10}[2]. Hence, from a collection of faulty ciphertexts $\{\tilde{C}_1, \tilde{C}_2, \ldots, \tilde{C}_n\}$, we predict the corresponding $\{\hat{Sak}_9^1, \hat{Sak}_9^2, \ldots, \hat{Sak}_9^n\}$, and use one of the distinguishers to discriminate the correct key byte.

It should be noted that the SEI distinguisher is useless in this context, as the distance to the uniform distribution will be the same for each hypothesis. Indeed, the inverse `SubBytes` and `AddRoundKey` operations only perform a permutation of the values of the end distribution, therefore keeping the numbers of occurrences as an invariant for every hypotheses. Thus, a substantial information about the distribution of the faulty value is necessary to successfully attack round 9.

3) Complexity analysis: As shown by equation 2, one can recover one byte of K_{10} by making hypotheses on itself, thus allowing to predict the corresponding byte of $\hat{Sak}_9^i$. Therefore, our attack needs a total of 2^8 hypotheses per key byte. Note that if the adversary is able to disturb only one byte of Sak_9, he has to perform 16 times the attack, each one targeting one byte index of K_{10}. On the contrary, if he perturbs the full `State`, the collection of faulty ciphertexts will allow him to retrieve all bytes of K_{10}.

4) Attack results: We compare the efficiency of our distinguishers against the considered fault models. The results are computed over 1000 simulations and summarized in Figure 1.

C. Attack on the 8*-th round*

1) Fault location: The fault is injected just after the ante-penultimate `AddRoundKey` operation of the `AES`, modifying Sak_8.

[2] A direct consequence is that a single fault perturbing the whole `State` can be seen as 16 faults, each one perturbing a single byte (indeed the bytes of the `State` are independent ones from the others in the last round).

	Max. likelihood	Min. mean HW
(a)	1	1
(b)	10	14
(c)	14	18

Figure 1. Attack on round 9: Number of required faults to retrieve (one byte of) K_{10} with a 99% probability.

2) Description of attack: We can express $\tilde{Sak}_8^i$ as a function of $\tilde{C^i}$, four bytes of K_{10} and one byte of $MC^{-1}(K_9)$:

$$\begin{aligned}\tilde{Sak}_8^i &= SB^{-1} \circ SR^{-1} \circ MC^{-1}(SB^{-1}\circ \\ &\quad SR^{-1}(\tilde{C^i} \oplus K_{10}) \oplus K_9) \\ &= SB^{-1} \circ SR^{-1}(MC^{-1}(SB^{-1}\circ \\ &\quad SR^{-1}(\tilde{C^i} \oplus K_{10})) \oplus MC^{-1}(K_9)).\end{aligned} \tag{3}$$

Each byte of $\tilde{Sak}_8^i$ can therefore be deduced using one hypothesis on four bytes of K_{10} and on a particular byte of $MC^{-1}(K_9)$. Moreover, an improvement can be made to enhance the efficiency of the attack using the SEI distinguisher. Indeed, as observed previously, the inverse SubBytes and the AddRoundKey operations can be omitted, allowing to mount this attack on $\tilde{Smod}_9$:

$$\tilde{Smod}_9^i = MC^{-1} \circ SB^{-1} \circ SR^{-1}(\tilde{C^i} \oplus K_{10}). \tag{4}$$

3) Complexity analysis: As shown by equation 4, one can recover four bytes of K_{10} by making hypotheses on their value, thus allowing to predict the corresponding byte of $\hat{Smod}_9^i$ (here four bytes of K_{10} are connected to one byte of $Smod_9^i$ due to inverse MixColumns). Therefore, our attack needs a total of 2^{32} hypotheses to retrieve 4 key bytes, and has to be performed four times to retrieve the entire K_{10}.

4) Attack results: Using the SEI distinguisher applied to the improved attack, the targeted four key bytes are retrieved with 99% probability using 6 faulty ciphertexts for the model a), using 14 faulty ciphertexts for the model b), and using around 80 faulty ciphertexts for the model c).

IV. Attacks on 7-th and 6-th round

In this section we study the resistance of AES to FA when a fault is inserted during rounds 6 or 7. As the fault occurs earlier in the encryption process, the bias it implies on the ciphertext is smaller and more difficult to exploit. Therefore, we use stronger fault models. Namely, we consider throughout this section *stuck at* faults, that is, perturbations such that the faulty byte(s) will take a constant but unknown value.

A. Attack on round 7

1) Fault model: In this section we assume that the adversary has the ability to inject a fault on a full *diagonal* of the State during several encryptions. By diagonal we mean a set of four bytes before SR that enter the same MC operation, *i.e.* bytes $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$ or $\{3, 4, 9, 14\}$.

This fault model could seem difficult to achieve in practice. However, in several implementations, the bytes are processed by 4-uples (for instance software implementations running on 32-bit CPUs; in such a case, a fault modifying the processing of one instruction can impact the four handled data bytes, see [11] for a practical example). Indeed, since bytes $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$ and $\{3, 4, 9, 14\}$ of the State Sak_r, Smc_r or Ssb_{r+1} correspond to the bytes that will be processed together during the next MixColumns operation, they are often manipulated together. Thus, a fault induced at such time will affect all four bytes.

In our model, the fault injection consists in setting the target part of the State to a constant (but potentially unknown) value. Our analysis shows that it does not need to be achieved with probability 1.

2) Description of the attack: We describe an attack resulting from the injection of a fault on a diagonal of Smc_7. Depending on which intermediate value of the State is the easiest to target, the same attack can be performed with a fault injection on Sak_7, Ssb_8 or Ssr_8 (in this last case, the fault should target a column of the State instead of a diagonal).

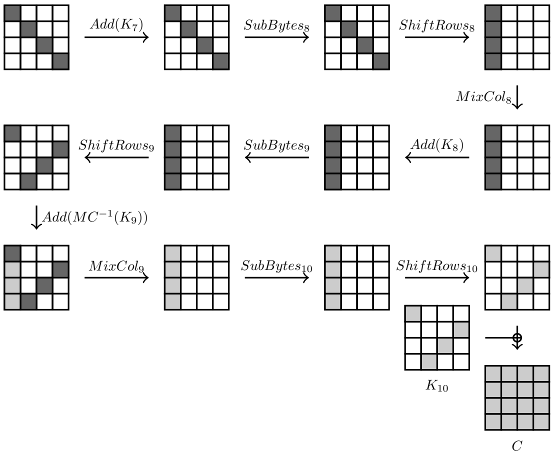

Figure 2. Sketch of the attack on round 7 using a fault on a diagonal of Smc_7. Dark bytes are set to a constant value due to the fault injection. Light bytes can be computed from the ciphertext and a guess on four bytes on K_{10}.

A sketch of the attack is depicted on Figure 2. It uses the following property: let us assume that an adversary manages to inject a fault on bytes $Smc_7^i[0, 5, 10, 15]$, during the encryption of ℓ different plaintexts, which means that these 4-uples of bytes take a constant value (over the ℓ values of i). In the remainder of this section, we denote such a phenomenon by *ℓ-multi-collision*. Figure 2 shows that this ℓ-multi-collision propagates to bytes $Smod_9^i[0, 7, 10, 13]$.

Let us consider a key guess $\hat{K}$. Then, starting from the ciphertext C^i and computing backwards, the resulting byte $\hat{S}mod_9^i[0]$ depends only on key bytes $\hat{K}_{10}[0, 7, 10, 13]$. For a time complexity of $\ell \cdot 2^{32}$ decryptions of the last round, we can then detect which values of these four bytes lead to a ℓ-multi-collision on byte $\hat{S}mod_9^i[0]$. A similar analysis enables the adversary to identify which values of 4-uples $\hat{K}_{10}[1, 4, 11, 14]$, $\hat{K}_{10}[2, 5, 8, 15]$ and $\hat{K}_{10}[3, 6, 9, 12]$ lead to ℓ-multi-collisions, respectively on byte $\hat{S}mod_9^i[7]$, $\hat{S}mod_9^i[10]$ and $\hat{S}mod_9^i[13]$.

3) Complexity analysis: Let us consider the first 4-uple of key bytes $\hat{K}_{10}[0, 7, 10, 13]$. We assume that for any wrong key guess, the bytes $\left(\hat{S}mod_9^i[0]\right)_{1 \le i \le \ell}$ are independent and uniformly distributed in $GF(256)$. Then, the probability that a wrong guess $\hat{K}_{10}[0, 7, 10, 13]$ leads to a ℓ-multi-collision is $2^{-8(\ell-1)}$. The average number of wrong guesses that cannot be discarded is then $2^{32-8(\ell-1)}$. For $\ell > 5$, only the right key guess passes the test with high probability.

The same analysis applies to the other 4-uples of key bytes. After exploiting the information obtained by faulting the encryptions, the adversary has to perform an exhaustive search on the remaining key candidates. Its estimated complexity is:

$$T_{search}(\ell, \ell) = \left(2^{32-8(\ell-1)}\right)^4 = 2^{128-32(\ell-1)} ,$$

which is practical for $\ell \ge 4$.

In the first step of the attack, one has to compute a part of the decryption for each of the ℓ ciphertexts in each of the 4 columns, under 2^{32} different key hypotheses. The complexity of this step is then $4\ell 2^{32}$. Therefore, the overall complexity of the attack is given by:

$$T_{attack}(\ell, \ell) = 2^{128-32(\ell-1)} + 4\ell 2^{32} .$$

4) Considering failed fault attempts: Let us consider now that the adversary can inject such a fault only with probability p. The attack described above still applies, although the multi-collision occurs only on ℓp of the ℓ values of $Smod_9^i[0, 7, 10, 13]$ (on average).

Therefore, after getting all the possible faulty ciphertexts, we have to select a threshold τ for the size of the largest multi-collision obtained with the right key guess. Instead of considering all key values that lead to an ℓ-multi-collision on byte $\hat{S}mod_9^i[0]$, we select the key guesses for which a τ-multi-collision occurs. As we wish to discard as many wrong keys as possible and to keep the right key in the selected pool with a good probability, $\tau \approx \ell p \ge 4$ is a natural choice.

The same modification applies to the other 4-uples of key bytes. Considering the fact that the τ-uple of indexes for which the collision occurs is the same for each of the bytes $Smod_9^i[0, 7, 10, 13]$, the adversary can perform an overall search on these τ-uples. This search only needs to be done once for all 4 quadruples of key bytes. The time complexity of the exhaustive search is now:

$$T_{search}(\ell, \tau) = \binom{\ell}{\tau} 2^{128-32(\ell-1)} .$$

Taking into account the first phase of the attack, the overall time complexity becomes:

$$T_{attack}(\ell, \tau) = \binom{\ell}{\tau} 2^{128-32(\ell-1)} + 4\ell 2^{32} .$$

5) Improving the complexity of the first step: In the description of our attack, the identification of the key candidates is performed by an exhaustive approach. Depending on the values of ℓ and τ, it might be improved by using an algorithm searching collisions in a list.

As the `MixColumns` operation is linear, so is its inverse. Therefore, $Smod_9[0]$ can be expressed as a linear function of $Sak_9[0,1,2,3]$. We denote by $L_{0,1}$ and $L_{2,3}$ the linear operations such that $Smod_9[0] = L_{0,1}(Sak_9[0,1]) \oplus L_{2,3}(Sak_9[2,3])$. We denote by $\lambda_{0,1}$ and $\lambda_{2,3}$ these values. Starting from the faulty ciphertext C^i, one can compute $\hat{\lambda}_{0,1} = L_{0,1}(\hat{Sak}_9^i[0,1])$ by guessing only $\hat{K}_{10}[0,13]$ and $\hat{\lambda}_{2,3} = L_{2,3}(\hat{Sak}_9^i[2,3])$ by guessing only $\hat{K}_{10}[7,10]$. We can then build two lists $\Lambda_{0,1}$ and $\Lambda_{2,3}$ of 2^{16} ℓ-uples of bytes. Each ℓ-uple contains the values of $\hat{\lambda}^i_{0,1}$ (or $\hat{\lambda}^i_{2,3}$) for $1 \leq i \leq \ell$.

Then, we guess which τ-uple of ciphertexts $(C^{i_1}, \ldots, C^{i_\tau})$ is the result of τ successful fault injections. For the right key guess, we have a τ-multi-collision on $\hat{Smod}_9^{i_1,\ldots,i_\tau}$, which can be expressed as:

$$\hat{\lambda}^{i_1}_{0,1} \oplus \hat{\lambda}^{i_1}_{2,3} = \ldots = \hat{\lambda}^{i_\tau}_{0,1} \oplus \hat{\lambda}^{i_\tau}_{2,3} ,$$

which is equivalent to:

$$\begin{aligned} \hat{\lambda}^{i_1}_{0,1} \oplus \hat{\lambda}^{i_2}_{0,1} &= \hat{\lambda}^{i_1}_{2,3} \oplus \hat{\lambda}^{i_2}_{2,3} \\ &\ldots \\ \hat{\lambda}^{i_1}_{0,1} \oplus \hat{\lambda}^{i_\tau}_{0,1} &= \hat{\lambda}^{i_1}_{2,3} \oplus \hat{\lambda}^{i_\tau}_{2,3} . \end{aligned}$$

In each ℓ-uple of $\Lambda_{0,1}$ (resp. $\Lambda_{2,3}$), we extract the τ corresponding bytes $\hat{\lambda}^{i_1,\ldots,i_\tau}_{0,1}$ (resp. $\hat{\lambda}^{i_1,\ldots,i_\tau}_{2,3}$, and keep the $\tau - 1$-uples $(\hat{\lambda}^{i_1}_{0,1} \oplus \hat{\lambda}^{i_2}_{0,1}, \ldots, \hat{\lambda}^{i_1}_{0,1} \oplus \hat{\lambda}^{i_\tau}_{0,1})$ (resp. .$\hat{\lambda}^{i_1}_{2,3} \oplus \hat{\lambda}^{i_2}_{2,3}, \ldots, \hat{\lambda}^{i_1}_{2,3} \oplus \hat{\lambda}^{i_\tau}_{2,3})$). This costs approximately $2^{16}\tau$ operations. Then, we sort both lists of 2^{16} $\tau - 1$-uples (for a cost of $2 \times 16 \times 2^{16}$) and enumerate all the collisions between both lists. In this step, the bottleneck is the search through both lists if the resulting list is expected to be smaller than the initial lists, or the construction of the resulting list if it is larger. As a consequence, the complexity of this step is $max\left(2^{16}, 2^{32-8(\tau-1)}\right)$.

Overall, by summing the complexities on the four columns and the possible guesses of $(i_1, \ldots, i_\tau)$, the complexity of this algorithm is:

$$\begin{aligned} T_{step1}(\ell, \tau) &= 8\ell 2^{16} + 4\tbinom{\ell}{\tau}\left(2^{16}\tau + 32 \times 2^{16} \right. \\ &\qquad \left. + \, max\left(2^{16}, 2^{32-8(\tau-1)}\right)\right) \\ &\approx 8\ell 2^{16} + 2^{24}\tbinom{\ell}{\tau} \end{aligned}$$

which is smaller than $4\ell 2^{32}$ for small values of ℓ and τ.

6) Results: The success probability of our attack, depending on p, ℓ and τ is given in Figure 3. This probability corresponds to the probability that the correct key will be present in the selected group of key candidates, *i.e.* we succeeded in at least τ of our fault injections. We also give the expected number of possible key candidates remaining after the first step. This directly gives us the complexity of the second step of our attack where all these key candidates must be tested in order to find the correct one. The complexity of the first step of the attack will be at most $4\ell 2^{32}$ but can be reduced for small values of ℓ and τ as shown above.

The table can be read as follows. With a probability of fault injection estimated at $p = 0.8$, the attacker may choose to fault 6 messages. Then, guessing the bits of the key as described above, he may look for 4-multi-collisions, that is a set of 4 messages among the 6 having the same value on the expected bytes. In that case, he will obtain a set of approximately $2^{35.9}$ possible keys. The correct key will belong in this set only if he has indeed obtained the expected fault 4 times among the 6 tries, that is with probability 0.9. If the adversary chooses instead to look for 5-multi-collision, he

Proba. of fault inj. (p)	Nb. of mess. (ℓ)	expected collision size (τ)	Success proba. of the attack	Nb. of key candidates
1	4	4	1	2^{32}
	5	5	1	1
0.9	4	4	0.66	2^{32}
	5	4	0.92	$2^{34.3}$
	6	4	0.98	$2^{35.9}$
	6	5	0.88	6
0.8	5	4	0.74	$2^{34.3}$
	6	4	0.90	$2^{35.9}$
	6	5	0.66	6
	8	5	0.94	56
0.7	6	4	0.75	$2^{35.9}$
	7	4	0.88	$2^{37.1}$
	7	5	0.65	21
0.5	8	4	0.64	$2^{38.1}$
	10	4	0.83	$2^{39.7}$
	10	5	0.62	252

Figure 3. Success probability of the attack on round 7 and estimated number of selected key candidates

will decrease the number of candidates to approximately 6 keys. However the correct key will be in this set only with probability 0.66. Thus, for a given p, there is a tradeoff between the number of messages to fault, the size of the multi-collision to search for, the success probability of the attack and the size of the set of possible remaining keys.

B. Attack on round 6

In this section, we show that there is still a possibly exploitable leak when the fault is injected at round 6. As it can be expected, the loss of security is less dramatic when the fault is injected earlier in the encryption process. But even if the attack depicted in this section is not easily applicable, we argue that it decreases the security level reached by some AES implementations. It should also be taken into account when choosing appropriate countermeasures for a given implementation.

1) Fault model: In this section we extend the diagonal fault model used in Section IV-A. We assume that we are able to fault 3 diagonals of the State Smc_6. As above, the fault consists in setting the corresponding bytes to an unknown fixed value with a probability p. This means that when the fault succeeds, only four bytes $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$, $\{2, 7, 8, 13\}$ or $\{3, 4, 9, 14\}$ keep their original values.

2) Description of the attack: A sketch of the attack is depicted in Figure 4. As previously, depending on which intermediate value of the State is more vulnerable to fault injections, the attack can be performed on Smc_6, Sak_6, Ssb_7 or Ssr_7 (in this last case, the fault should target 3 columns of the State instead of 3 diagonals).

As before, we assume that the fault injections during several encryptions result in a τ-multi-collision on the dark grey bytes in Figure 4. This multi-collision propagates to 12 bytes of the State Ssr_8. After the MixColumns operation of round 8, the first column of State Smc_8 will have only 2^8 possible values. Thus, we can expect a collision between two such columns with only 2^4 successfully faulty ciphertexts. This new collision is depicted in grey on Figure 4. It propagates to bytes $Smod_9[0, 7, 10, 13]$. This collision can be detected, as in the previous attack, by guessing 4 bytes of K_{10}.

3) Complexity analysis: Let us first assume that we have a probability 1 of fault injection on the three diagonals. According to [14], the expected number of messages needed to obtain a τ-multi-collision from a set of size r is given by:

$$N(r, \tau) \sim_{r\to\infty} \sqrt[\tau]{\tau!}\,\Gamma\left(1 + \frac{1}{\tau}\right) r^{1-1/\tau},$$

where Γ is the gamma function $\Gamma(x) = \int_0^\infty e^{-t}t^{x-1}dt$.

According to this formula, a 2-multi-collision will occur on the first column of State Smc_8 after approximately $\ell = 20$ ciphertexts have been successfully faulted. A 3-multi-collision will occur after $\ell = 65$ ciphertexts, a 4-multi-collision after $\ell = 128$ and a 5-multi-collision will appear after $\ell = 202$ ciphertexts. These numbers could be increased to get a better success probability (see results presented in Figure 5 for possible choices).

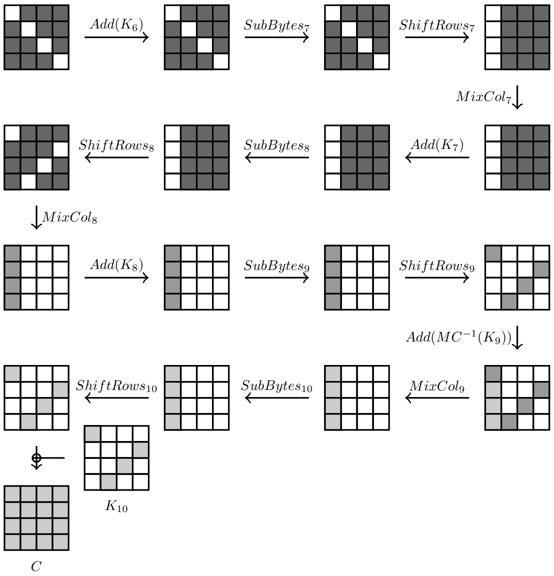

Figure 4. Sketch of the attack on round 6 using a fault on Smc_6. Dark bytes are set to a constant value due to the fault injection. Light bytes can be computed from the ciphertexts and a guess of four bytes on K_{10}. Collision on the intermediate grey bytes can be obtained with 2^4 faulty ciphertexts.

When the probability p of obtaining a faulty ciphertext is less than 1, the number of messages given above should be divided by p in order to obtain the expected multi-collision even when the fault is not certain.

The remainder of the attack proceeds as previously. The number of key candidates after the first step of the attack is:

$$T_{search}(\ell,\tau) = \binom{\ell}{\tau} 2^{128-32(\ell-1)} \ .$$

Taking into account the first phase of the attack, the overall time complexity is about:

$$T_{attack}(\ell,\tau) = \binom{\ell}{\tau} 2^{128-32(\ell-1)} + 4\ell 2^{32} \ .$$

Finally, the Figure 5 presents the success probability and the remaining key search complexity of our attack for several values of p, ℓ and τ.

Proba. of fault inj. (p)	Nb. of mess. (ℓ)	expected collision size (τ)	Success proba. of the attack	Nb. of key candidates
1	128	4	0.36	$2^{55.3}$
	155	4	0.59	$2^{56.5}$
	202	5	0.28	$2^{31.3}$
	245	5	0.54	$2^{32.7}$
0.9	143	4	0.36	2^{56}
	174	4	0.60	$2^{57.1}$
	225	5	0.28	2^{32}
	273	5	0.54	$2^{33.5}$
0.8	160	4	0.36	$2^{56.6}$
	194	4	0.59	$2^{57.8}$
	253	5	0.29	2^{33}
	307	5	0.54	$2^{34.4}$
0.7	183	4	0.36	$2^{57.4}$
	222	4	0.59	$2^{58.6}$
	289	5	0.29	$2^{33.9}$
	350	5	0.54	$2^{35.3}$
0.5	256	4	0.36	$2^{59.4}$
	310	4	0.59	$2^{60.5}$
	404	5	0.29	$2^{36.3}$
	490	5	0.54	$2^{37.7}$

Figure 5. Success probability of the attack on round 6 and number of selected key candidates

V. Conclusion

In this work we have described several fault attacks on AES-128 that lead to the recovery of the whole secret key. Unlike previous fault attacks, these attacks are in a faulty ciphertexts only model. That is, the adversary does not know nor control the encrypted messages. In particular he cannot obtain a pair of correct/faulty ciphertexts from a same plaintext.

Depending on the disturbed round, we use different fault models. The common point of the models we studied is that the distribution of the faulty bytes is non uniform. This can be achieved when the faulty value is forced to an unknown constant or, more loosely, when the effect of the fault can be modelized as an AND or OR operation with an unknown error byte.

Our attacks require a reasonable number of faulty ciphertexts (a few hundreds for the worse attack), which makes them practical. We also tolerate failed injections and give results depending on the success probability of obtaining the expected fault. Our results based on simulations show that such attacks present a real threat to cryptographic algorithms and that FA countermeasures should not rely exclusively on input randomization (like the scheme proposed in [13]), but should also use dedicated countermeasures (redundancy, infection . . .).

Acknowledgment

The authors would like to thank Ryad Benadjila for his helpful tricks improving the efficiency of our attack simulations.

References

[1] EMV Book 2 - Security and Key Management. EMVCo website, 2011.

[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. In L. Breveglieri and I. Koren, editors, *Workshop on Fault Diagnosis and Tolerance in Cryptography – FDTC'04*, pages 330–342. IEEE Computer Society, 2004.

[3] Guido Bertoni and Benedikt Gierlichs, editors. *2012 Workshop on Fault Diagnosis and Tolerance in Cryptography, Leuven, Belgium, September 9, 2012*. IEEE, 2012.

[4] I. Biehl, B. Meyer, and V. Müller. Differential Fault Analysis on Elliptic Curve Cryptosystems. In M. Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 131–146. Springer, 2000.

[5] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystem. In B.S. Kalisky Jr., editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 1997.

[6] J. Blömer and V. Krummel. Fault Based Collision Attacks on AES. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC 2006*, volume 4236 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2006.

[7] D. Boneh, R.A. DeMillo, and R.J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults. In W. Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 1997.

[8] Christophe Clavier and Kris Gaj, editors. *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*. Springer, 2009.

[9] Jean-Sébastien Coron, Antoine Joux, Ilya Kizhvatov, David Naccache, and Pascal Paillier. Fault Attacks on RSA Signatures with Partially Unknown Messages. In Clavier and Gaj [8], pages 444–456.

[10] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Fault Attacks Against EMV Signatures. In Josef Pieprzyk, editor, *CT-RSA*, volume 5985 of *Lecture Notes in Computer Science*, pages 208–220. Springer, 2010.

[11] Amine Dehbaoui, Amir-Pasha Mirbaha, Nicolas Moro, Jean-Max Dutertre, and Assia Tria. Electromagnetic Glitch on the AES Round Counter. In *COSADE*, 2013.

[12] FIPS PUB 197. *Advanced Encryption Standard*. National Institute of Standards and Technology, November 2001.

[13] Sylvain Guilley, Laurent Sauvage, Jean-Luc Danger, and Nidhal Selmane. Fault Injection Resilience. In Luca Breveglieri, Marc Joye, Israel Koren, David Naccache, and Ingrid Verbauwhede, editors, *FDTC*, pages 51–65. IEEE Computer Society, 2010.

[14] M. S. Klamkin and D. J. Newman. Extensions of the Birthday Surprise. *Journal of Combinatorial Theory*, 3:279–282, 1967.

[15] Ronan Lashermes, Guillaume Reymond, Jean-Max Dutertre, Jacques Fournier, Bruno Robisson, and Assia Tria. A DFA on AES Based on the Entropy of Error Distributions. In Bertoni and Gierlichs [3], pages 34–43.

[16] Philippe Maurine. Techniques for EM Fault Injection: Equipments and Experimental Results. In Bertoni and Gierlichs [3], pages 3–4.

[17] G. Piret and J.-J. Quisquater. A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In C.D. Walter, Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2003.

[18] Matthieu Rivain. Differential Fault Analysis on DES Middle Rounds. In Clavier and Gaj [8], pages 457–469.

[19] Sergei Skorobogatov and Ross Anderson. Optical Fault Induction Attack. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 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):967–970, 2000.