Complete Reverse-Engineering of AES-like Block Ciphers by SCARE and FIRE Attacks

Christophe Clavier, Quentin Isorez,
Damien Marion, and Antoine Wurcker

XLIM-CNRS, Université de Limoges, Limoges, France
christophe.clavier@unilim.fr
quentin.isorez@etu.unilim.fr
damien.marion@unilim.fr
antoine.wurcker@xlim.fr

Abstract. Despite Kerckhoffs's principle, proprietary or otherwise secret cryptographic algorithms are still used in real life. For security and efficiency reasons a common design practice simply modifies some parameters of widely used and well studied encryption standards. In this paper, we investigate the feasibility of reverse engineering the secret specifications of an AES-like block cipher by a FIRE attack based on Ineffective Fault Analysis (IFA) or by SCARE techniques based on two models of collision power analysis. In the considered fault or observational models, we demonstrate that an adversary who does not know the secret key can recover the full set of secret parameters of an AES-like software implementation and, in some models, even if it is protected by common Boolean masking and shuffling of independent operations. We thereby intend to demonstrate that protecting the implementation of such AES-like function is not an option even if its specifications are not public.

Keywords: embedded devices, reverse engineering, AES, side-channel analysis, SCARE, fault attack, FIRE

1 Introduction

Despite Kerckhoffs's principle, the security is sometimes expected through obscurity, and proprietary or otherwise secret cryptographic algorithms are still used in civil applications such as GSM or Pay-TV systems and for diplomatic or military usages. For sake of simplicity a strategy for designing a secret encryption function may consist in only modifying some parameters of a well studied and widely used public function. Doing this, the design and development costs are reduced, and provided that the secret parameters are carefully chosen the designer can expect inheriting the strength of the public function structure like the Substitution Permutation Network in the exemplary case of AES. In this paper we focus on the problem of recovering – by means of physical attacks – the parameters of a secret AES-like block cipher defined as a modified AES where some or all of its parameters are replaced with secret non-standard values. We present three contributions – different attacks under different models – which intent to show that a broad range of methods may be used to that end. We give here short introductions to fault analysis and side-channel analysis to introduce our three different contributions.

Fault Analysis Fault analysis on embedded systems like smartcards has first been introduced in late 1996 [1, 5] to threaten the two most famous cryptographic algorithms that were in use at that time: DES and RSA. Since then many other contributions have been published that also describe how to recover secret cryptographic keys by inducing faults that produce erroneous results on a variety of public cryptographic algorithms. Interestingly – and contrarily to the emblematic Differential Fault Analysis (DFA) that derives information about the key from differences between a genuine and a faulted outputs – some variants of fault analysis can exploit situations where a fault attempt resulted in no modification of the normal output. This is the case of Safe Errors Analysis (SEA) [33, 16] that considers faults whose effect on intermediate data has no influence on the outcome of the execution, as well as Ineffective Fault Analysis (IFA) where faults having no effect at all are yet informative [7, 10].

A first contribution of this paper, already published in [9], applies the ineffective fault analysis concept to devise a FIRE[1] attack on a secret AES-like block cipher. There is little literature about reverse engineering by means of fault analysis and, as far as we know, this is the first published FIRE attack that applies on such AES-like function. Under the byte stuck-at-0 fault model, our attack exploits the effective or ineffective nature of faults injected when reading an S-Box value from memory. We show that an unprotected implementation of an AES-like block cipher is vulnerable to our FIRE attack that can recover the full set of secret parameters. Beside providing new techniques for recovering AES parameters by means of IFA, our attack demonstrates the necessity to protect the implementation of a block cipher against fault attacks even though its specifications are not public.

Side Channel Analysis Side Channel Analysis (SCA) has been introduced by Kocher et al. [18, 19] as an efficient means to recover the secret key of cryptographic algorithms on embedded devices. Since then many new attack techniques have been discovered and new efficient countermeasures against these attacks have been proposed. Most of this research activity concerned the classical context of key recovery where the target encryption function is publicly known as in the emblematic cases of DES [23] and AES [24] standards.

It is thus of interest to study to which extent SCA techniques – essentially through power or electromagnetic measurements – can be used to recover the structure details and/or the parameter values of an encryption algorithm whose specifications are kept secret. Novak [25] has first described a SCARE[2] technique that reveals the content of one of the two substitution tables used in an authentication and key agreement secret GSM algorithm. His attack considers the observational model where the adversary is able to decide whether two instances of precisely located intermediate data collide or not – without identifying their values – based on the side-channel observation of their execution trace(s). In the same model, Clavier [8] later improved this attack by revealing both S-Boxes without prior knowledge of the encryption key. In [13] Daudigny et al. proposed a SCARE of the standard DES. They used DPA to infer so-called *scheduling information* which reveals when particular bits are manipulated, from which the suppos-

[1] FIRE: Fault Injection for Reverse Engineering.
[2] SCARE: Side-Channel Analysis for Reverse Engineering.

edly unknown permutation functions are derived. Two other works extended the usage of SCARE to whole classes of ciphers sharing the same structure: Real et al. [29] first revealed the round function of any unknown hardware Feistel implementation while Rivain et al. show in [30] how to exploit S-Boxes collisions to retrieve an equivalent representation of any secret Substitution-Permutation Network based encryption function. Other publications [14, 26] also deal with side-channel analysis to reveal information about secret algorithms.

The two other contributions of this paper investigate the feasibility of reverse engineering the secret specifications of an AES-like block cipher by means of two different models of collision power analysis:

First we consider the collision of values model where the attacker is able to detect whether the two couples $(x, y = S(x))$ and $(x', y' = S(x'))$ of input/output bytes of two different S-Box computations (or readings) are identical or not. This study, already published in [12], demonstrates that an adversary who does not know the secret key can efficiently recover the full set of secret parameters of an AES-like software implementation even if it is protected by a common set of countermeasures. While our work has much in common with [30] (same attacker model, same attacker goal) they still have distinct contributions. On one hand Rivain et al. recover any SPN-based function while our method only applies to AES with secret parameters. On the other hand they assume an unprotected implementation while we describe a variant of our attack that deals with some classical side-channel countermeasures.

It may be argued that distinguishing between collision and non-collision of (couples of) values by observing the side-channel leakage may be a quite challenging task in practice. This is why we thought interesting to investigate a more relaxed observational model where the adversary only identifies collisions and non collisions of couples of Hamming weights. As the power consumption is classically modelized as a linear function of the Hamming weight of the manipulated data, we see the ability to detect collisions of Hamming weights – $(\mathrm{HW}(x), \mathrm{HW}(y)) = (\mathrm{HW}(x'), \mathrm{HW}(y'))$ – as a weaker and more reasonable attacker model. While these collisions are less informative than collisions of value, we still devised a method – this is the novel contribution of this paper – that exploits them to retrieve the secret parameters without knowing the key. After presenting an attack in the unprotected implementation scenario, we explain how to adapt it so that it still applies in the presence of one of the two countermeasures considered in the collision of values model.

Organisation of the Paper The paper is organized as follow: Section 2 briefly describes the AES and defines what we call an AES-like function which is the target of our attacks. We present the considered attacker models in Section 3 and analyse how they are related. In Section 4 we explain step by step how to recover the full set of parameters of the AES-like by a FIRE method. In Section 5 we show how to achieve the same goal by a SCARE attack under the collision of values model both on unprotected and protected implementation. In Section 6 we present an original work that also devise a SCARE attack though in the relaxed model of collision of Hamming weights. We discuss possible countermeasures against our attacks in Section 7 and conclude the paper in Section 8.

2 AES-like Block Cipher

We give hereafter a brief description of the 128-bit version of AES, and define what we mean in this paper by an AES-like block cipher. For more precise information about the AES we refer the reader to the NIST standard [24] which includes its full specifications.

2.1 Description of the AES

In the process of AES computation, a byte is considered as an element of the finite field $GF(2^8)$, and each 16-byte internal state may be represented as a square 4×4 matrix. The mapping between the vector and matrix representations is done by numbering the elements column by column as described on Figure 1.

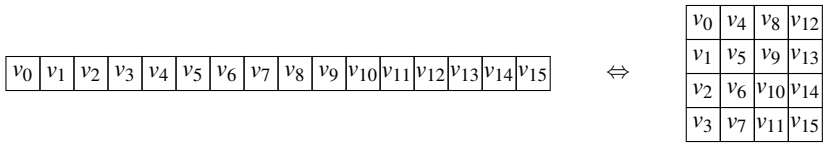

Fig. 1. Mapping between vector and matrix representations in AES

Given a 128-bit plaintext $M = (m_0, \ldots, m_{15})$ and a 128-bit ciphering key $K = (k_0, \ldots, k_{15})$ the AES computes a 128-bit ciphertext $C = (c_0, \ldots, c_{15})$, as depicted in Figure 2, by first XOR-ing M with K and then updating the result state S_0 through 10 rounds. For $r = 1, \ldots, 9$, each AES round number r computes its output state S_r by successively applying four transformations `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey` to its input S_{r-1}. The ciphertext is finally defined as the output of a 10^{th} and last round which does not include the `MixColumns` operation. The encryption process is summarized as follow:

$$
\begin{aligned}
S_0 &\leftarrow M \oplus K_0 \qquad (K_0 = K) \\
S_r &\leftarrow \texttt{MixColumns}(\texttt{ShiftRows}(\texttt{SubBytes}(S_{r-1}))) \oplus K_r \qquad (r = 1, \ldots, 9) \\
C &\leftarrow \texttt{ShiftRows}(\texttt{SubBytes}(S_9)) \oplus K_{10}
\end{aligned}
$$

The `SubBytes` transformation is a permutation over $GF(2^8)$ defined by an S-Box table S. The `ShiftRows` consists in rotating each row number i ($i = 0, \ldots, 3$) by i bytes to the left. The `MixColumns` computes each column of its output as the product of a constant matrix by the corresponding input column. Finally the `AddRoundKey` computes the XOR (addition in $GF(2^8)$) between the current state and the round key K_r. The different round keys K_r involved in the encryption process are derived from K through the key scheduling function depicted on Figure 3. The `RotWord` operation simply rotates each byte of a column by one position upward, `SubWord` applies the S-Box to each byte of the column, and `Rcon`$[r]$ is a round dependent constant word equal to $(\rho^{r-1}, 0, 0, 0)$ with $\rho = 2$.

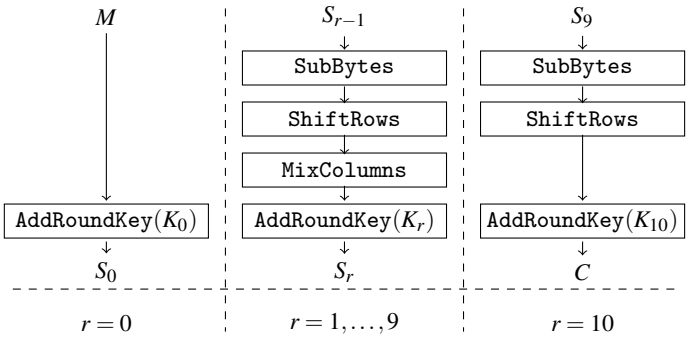

Fig. 2. The AES encryption path

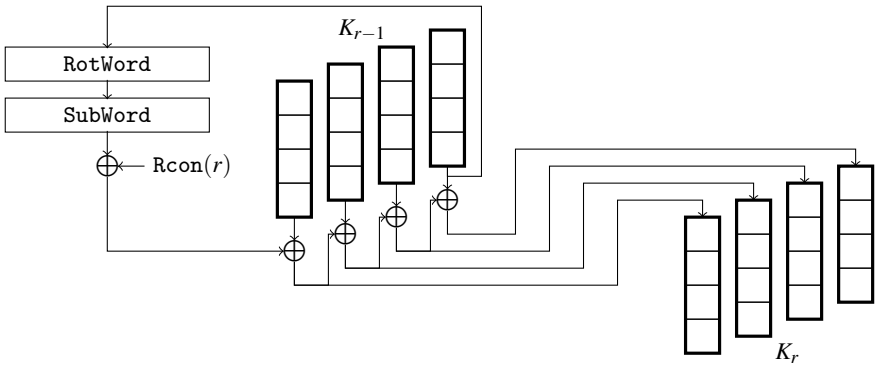

Fig. 3. The AES key schedule

2.2 Definition of an AES-like Block Cipher

From the definition of the 128-bit AES it is possible to derive a large class of encryption functions which differ from the standard AES while preserving its essential structure of a Substitution Permutation Network as well as the number and width of its internal values.

We define an AES-like block cipher as any function which is derived from AES by modifications of the following parameters:

1. the S-Box table S can be replaced by any other one that preserves the property that the `SubBytes` function is a permutation over $GF(2^8)$ elements,
2. in the `ShiftRows` transformation each row number i is rotated by σ_i bytes to the left, where σ_i can be any value from 0 to 3 (note that $\sigma_i = i$ for the standard AES),
3. the constant matrix which defines the `MixColumns` operation can be replaced by different configuration depending if we consider the FIRE or the SCARE setting:
   3.1. FIRE attack: any circulant matrix based on a 4-tuple $(\gamma_0, \ldots, \gamma_3)$ of $GF(2^8) \setminus \{0\}$,
   3.2. SCARE attacks: any 16-tuple $(\alpha_0, \ldots, \alpha_{15})$ of $GF(2^8)$,
4. the `RotWord` operation in the key schedule rotates the column by η bytes upward, where η can be any value from 0 to 3 (note that $\eta = 1$ for the standard AES),
5. the round dependent constant word `Rcon`$[r]$ involved in the key schedule for the computation of K_r is defined as $(\rho^{r-1}, 0, 0, 0)$ where ρ can take any non-zero byte value (note that $\rho = 2$ for the standard AES).

Figures 4 to 8 depict the possible degrees of freedom on the parameters of `ShiftRows`, `MixColumns`, `RotWord` and `Rcon` respectively.

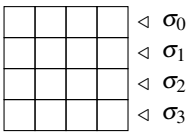

Fig. 4. `ShiftRows` parameters

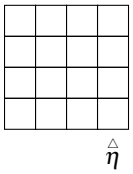

Fig. 5. `RotWord` parameter

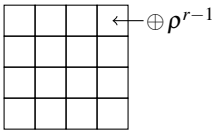

Fig. 6. `Rcon`$[r]$ parameter

$$\begin{bmatrix} \gamma_0 & \gamma_1 & \gamma_2 & \gamma_3 \\ \gamma_3 & \gamma_0 & \gamma_1 & \gamma_2 \\ \gamma_2 & \gamma_3 & \gamma_0 & \gamma_1 \\ \gamma_1 & \gamma_2 & \gamma_3 & \gamma_0 \end{bmatrix}$$

Fig. 7. `MixColumns` matrix (FIRE)

$$\begin{bmatrix} \alpha_0 & \alpha_4 & \alpha_8 & \alpha_{12} \\ \alpha_1 & \alpha_5 & \alpha_9 & \alpha_{13} \\ \alpha_2 & \alpha_6 & \alpha_{10} & \alpha_{14} \\ \alpha_3 & \alpha_7 & \alpha_{11} & \alpha_{15} \end{bmatrix}$$

Fig. 8. `MixColumns` matrix (SCARE)

For sake of simplicity, in the following sections, we shall simply denote by *AES* the secret AES-like function that the attacker aims at reverse-engineer, and we shall refer to the standard AES function as *standard AES*.

2.3 Notations

We introduce the following further notations:

1. Global notations:
   - $K_r = (k_{r,0}, \ldots, k_{r,15})$, the r^{th} round key (for $r = 0, \ldots, 10$)
   - $X_r = (x_{r,0}, \ldots, x_{r,15})$, the `SubBytes` input of round r (for $r = 1, \ldots, 10$)
   - $Y_r = (y_{r,0}, \ldots, y_{r,15})$, the `SubBytes` output of round r (for $r = 1, \ldots, 10$)
   - $Z_r = (z_{r,0}, \ldots, z_{r,15})$, the `MixColumns` input of round r (for $r = 1, \ldots, 9$)
   - $T_r = (t_{r,0}, \ldots, t_{r,15})$, the `MixColumns` output of round r (for $r = 1, \ldots, 9$)
2. notations used in Sect. 4 (FIRE attack):
   - $\lambda_i = k_{0,i} \oplus S^{-1}(0)$ (for $i = 0, \ldots, 15$)
   - $mk_{i,j}$ the byte value verifying equation: $\gamma_j * S(mk_{i,j} \oplus k_{0,0}) = k_{1,i} \oplus S^{-1}(0)$ (for $i = 0, \ldots, 15$ and $j = 0, \ldots, 3$)
   - $\beta_{i,j} = \gamma_i / \gamma_j$ (for $i, j = 0, \ldots, 3$)
3. notations used in Sect. 5 and 6 (SCARE attacks):
   - $\mu_{r,i,j} = k_{r,i} \oplus k_{r,j}$ (for $r = 0, \ldots, 10$ and $i, j = 0, \ldots, 15$)

3 Attacker Models

We consider the chosen plaintext scenario where the attacker owns a device (e.g. a smartcard) embedding a software implementation of a secret AES-like encryption function. He can query the device with chosen plaintexts and receives the corresponding ciphertexts. He is assumed to ignore the value of the key K and his goal is to reverse-engineer all the secret parameters of the encryption function by analysing faults effects for the FIRE attack, or the side-channel traces of each encryption for the SCARE attacks.

It is obvious that the cryptographic strength of an AES-like block cipher defined in Section 2.2 may range from very weak to reasonably strong functions. Even, probably a quite small fraction of them can be acceptable for a safe cryptographic usage. Nevertheless, as a conservative option, we choose to consider a blind attacker who does not disqualify a possible function regarding the relevance of its parameters, but rather accepts a priori any set of parameters – $(S, \{\sigma_i\}_i, \{\gamma_i\}_i, \eta, \rho)$ for FIRE model and $(S, \{\sigma_i\}_i, \{\alpha_i\}_i, \eta, \rho)$ for SCARE models – modifiable according to our definition.

3.1 FIRE Attack Model: Byte Stuck at Zero

For the FIRE attack described in Section 4 we assume a byte stuck at 0 fault model when reading an S-Box output from memory. That means that when a fault is injected during the reading of a targeted S-Box, the retrieved value is forced to zero whatever its original value. In this model, the attacker can infer whether the original (non faulted) value of the S-Box output is equal to zero or not by encrypting the same plaintext twice: once without any fault and once with a fault attempt. As depicted on Figures 9 and 10, the fault attempt has no local effect (case of an ineffective fault) if the original value is already equal to zero, whereas it actually produces a modification of the read S-Box

output if its original value is different from zero. In the former case an IFA occurs which is identified by the fact that both resulting ciphertexts are equal, while they are different in the later case.

Notice that in the particular case of the dual-execution countermeasure the IFA can be detected with only the faulted execution. Indeed this countermeasure executes the cipher process twice and does not return any output in case of difference between the two ciphertexts. So an effective fault on one branch will be detected and the card will refuse to output a value whereas an ineffective one will let both branches unchanged and the card will output the value. The fact that the device agrees to output something or not is the binary information for the detection of an IFA.

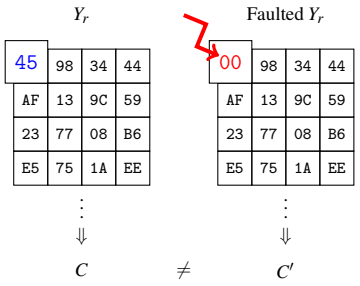

Fig. 9. Example of IFA no-occurrence

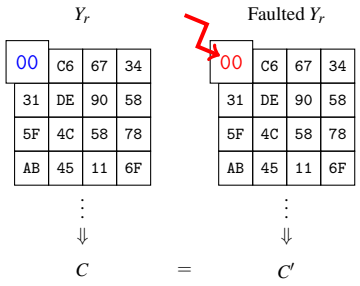

Fig. 10. Example of IFA occurrence

3.2 SCARE Attack Models: S-Box Collision

For the first SCARE attacks detailed in Section 5 we make the observational assumption that the attacker can identify S-Box collisions of values by side-channel analysis. More precisely, given two power trace segments T and T' corresponding to two table lookups $y = S(x)$ and $y' = S(x')$ in the AES computation[3], the attacker can decide whether $(x,y) = (x',y')$ or not, based on a side-channel distinguisher. This side-channel collision of values model has already been assumed in many key recovery attacks [31, 32, 2, 3, 4, 11] as well as for reverse-engineering purpose [25, 8]. Notably Bogdanov [4] used exactly the same model as ours applied on AES S-Boxes.

It may be argued that detecting collisions between two S-Boxes based on traces from two different executions is more difficult than from a unique trace and may result in less reliable decisions due to possible differences in the experimental conditions (temperature,...). As we think that this is a debatable question, we choose to present this attack in both settings: the *inter-traces* scenario where the attacker can detect collisions from different traces, and the *intra-trace* one where he can not.

For the SCARE attack presented in Section 6 we adopt the relaxed model of S-Box collisions of Hamming weights. Instead of identify whether $(x,y) = (x',y')$ the

[3] The two S-Box lookups may be located at different rounds, and possibly on different traces with different plaintexts.

attacker is merely required to decide whether $(\mathrm{HW}(x),\mathrm{HW}(y)) = (\mathrm{HW}(x'),\mathrm{HW}(y'))$. The choice of this relaxed model is supported by the underlying physical behaviour of an 8-bit micro-processor where the eight data bus lines contribute in a supposedly additive manner to the consumption. This obviously suggests the Hamming weight leakage model which has proved to be relevant and efficient particularly for several Simple Power Analysis [20, 21, 22] and for Correlation Power Analysis [6].

4 FIRE Attack using IFA

In this section we describe step by step how to recover the secret parameters of an AES implementation that does not feature any side-channel countermeasure. The assumed fault model introduced in Sect. 3.1 allows the attacker to detect by IFA if $y_{r,i} = S(x_{r,i}) = 0$ or not when he induces a fault on the computation or reading of S-Box i at round r.

We remind the reader that the method detailed below was already published in [12], this version offers more effort of clarity.

Retrieving the coefficients of the `MixColumns` matrix involves the multiplicative orders of the $\beta_{i,j} = \gamma_i/\gamma_j$ elements. We make the assumption that at least one of them has order 255 (which means that it generates $GF(2^8)\backslash\{0\}$). We have counted that this condition is verified for 95.28% of all 255^4 possible quadruplets.

We now introduce some preliminary results used in the description of the attack.

Lemma 1. *The knowledge of $(\lambda_0,\ldots,\lambda_{15})$ allows to select plaintext bytes m_i and m_j such that $x_{1,i} = x_{1,j}$ (for all $i,j = 0,\ldots,15$).*

Proof. For any m_i we have $x_{1,i} = m_i \oplus k_{0,i}$. Selecting $m_j = m_i \oplus \lambda_i \oplus \lambda_j$ induces that:

$$\begin{aligned} x_{1,j} &= m_i \oplus \lambda_i \oplus \lambda_j \oplus k_{0,j} \\ &= m_i \oplus k_{0,i} \oplus S^{-1}(0) \ \oplus k_{0,j} \oplus S^{-1}(0) \ \oplus k_{0,j} \\ &= m_i \oplus k_{0,i} \\ &= x_{1,i} \end{aligned}$$

Corollary 1. *The knowledge of $(\lambda_0,\ldots,\lambda_{15})$ and `ShiftRows` parameters allows the attacker to retrieve any value $mk_{i,j}$.*

Proof. By choosing $m_k = \lambda_k$ one can force to zero the output of the corresponding S-Box. Knowing `ShiftRows` parameters, an attacker is thus able to build a plaintext that induces three zero values at chosen positions in an arbitrary input column of `MixColumns`. He then exhausts the value of the last byte of this column by modifying the corresponding plaintext byte, and wait until an IFA occurs on a related chosen S-Box of second round. For chosen i and j, where i is the index of the byte implied in the IFA at line $\ell = i \bmod 4$ and column $c = \lfloor i/4 \rfloor$, and where γ_j is the active `MixColumns` coefficient, the position of the active plaintext byte expresses as $i' = \ell' + 4c'$ where $\ell' = (\ell + j) \bmod 4$ and $c' = (c + \sigma_{\ell'}) \bmod 4$, and we have:

$$
\begin{aligned}
y_{2,i} &= 0 \\
x_{2,i} &= S^{-1}(0) \\
\gamma_j * S(m_{i'} \oplus k_{0,i'}) \oplus k_{1,i} &= S^{-1}(0) \\
\gamma_j * S(m_{i'} \oplus \lambda_0 \oplus \lambda_0 \oplus k_{0,i'}) &= k_{1,i} \oplus S^{-1}(0) \\
\gamma_j * S((m_{i'} \oplus \lambda_0 \oplus \lambda_{i'}) \oplus k_{0,0}) &= k_{1,i} \oplus S^{-1}(0)
\end{aligned}
$$

from which the definition of $mk_{i,j}$ induces that $mk_{i,j} = m_{i'} \oplus \lambda_0 \oplus \lambda_{i'}$.

Lemma 2. *For $i \in \{0,4,1,5,2,6,3,7\}$, we have $k_{1,i} \oplus k_{1,i+8} = \lambda_{i+4} \oplus \lambda_{i+8}$.*

Proof.

$$
\left\{
\begin{array}{l}
k_{1,i+4} = k_{1,i} \oplus k_{0,i+4} \\
k_{1,i+8} = k_{1,i+4} \oplus k_{0,i+8}
\end{array}
\right\}
$$

$$
\Downarrow
$$

$$
k_{1,i} \oplus k_{1,i+8} = k_{0,i+4} \oplus k_{0,i+8} = \lambda_{i+4} \oplus \lambda_{i+8}
$$

4.1 Retrieving K_0 up to a Constant Byte

In order to retrieve $k_{0,i}$ up to the constant value $S^{-1}(0)$ we fault the output of the i^{th} S-Box of the first round while exhausting m_i. When an IFA occurs, we have $y_{1,i} = 0$ from which we deduce that $m_i \oplus k_{0,i} = S^{-1}(0)$. This particular value of m_i is equal to $\lambda_i = k_{0,i} \oplus S^{-1}(0)$. Looping on the plaintext byte position i we can recover $(\lambda_0, \ldots, \lambda_{15})$ which is K_0 up to a XOR with $S^{-1}(0)$.

4.2 Retrieving `ShiftRows` Parameters

To retrieve the `ShiftRows` parameters we first manage to obtain a reference plaintext that produces an IFA on the first S-Box of the second round. To that end we successively encrypt plaintexts where bytes m_0, m_4, m_8 and m_{12} are simultaneously exhausted. When the IFA occurs we know that $y_{2,0} = 0$. This byte depends on four plaintext bytes, one on each row of the state matrix, and positions of those bytes are directly related to the `ShiftRows` parameters. We can identify the positions of these four bytes by noticing that changing any of them modifies $y_{2,0}$ and breaks the IFA, whereas changing any of the 12 other ones lets $y_{2,0}$ unmodified. By encrypting the reference plaintext with only one byte modified each time we obtain the four positions that remove the IFA. Each one of these positions reveals a `ShiftRows` parameter. Indeed, if changing m_{r+4c} removes the IFA on $y_{2,0}$ it implies that this element, which belongs to row r, is shifted from column c to column 0 by `ShiftRows`. We thus infer that $\sigma_r = c$.

4.3 Reducing `MixColumns` Entropy by Retrieving $\beta_{i,j}$ Orders

To obtain the multiplicative orders of the $\beta_{i,j}$ quotients we target successive IFAs on the first S-Box of the second round. Without loss of generality, we first explain how to learn the order of $\beta_{1,2}$. Note that an IFA on $y_{2,0}$ implies the following equation:

$$\gamma_0 * z_{1,0} \oplus \gamma_1 * z_{1,1} \oplus \gamma_2 * z_{1,2} \oplus \gamma_3 * z_{1,3} \oplus k_{1,0} = S^{-1}(0)$$

This equation can be simplified by fixing $m_{4\sigma_0}$ to the value $mk_{0,0} \oplus (\lambda_0 \oplus \lambda_{4\sigma_0})$ where $mk_{0,0}$ is determined according to Corollary 1. This implies that $\gamma_0 * z_{1,0} = k_{1,0} \oplus S^{-1}(0)$ and the IFA equation thus reduces to:

$$\gamma_1 * z_{1,1} \oplus \gamma_2 * z_{1,2} \oplus \gamma_3 * z_{1,3} = 0$$

As a second trick, we also force $z_{1,3} = 0$ by fixing $m_{3+4\sigma_3}$ to $\lambda_{3+4\sigma_3}$. The IFA equation becomes:

$$\gamma_1 * z_{1,1} \oplus \gamma_2 * z_{1,2} = 0$$

At this point we enter an iterative process. At each step $k \geqslant 0$ the unknown value of $z_{1,1}$ is fixed and denoted by $\tau^{(k)}$ while we exhaust the unknown value of $z_{1,2}$ (actually by exhausting $m_{2+4\sigma_2}$) until an IFA occurs. This $z_{1,2}$ value will replace $z_{1,1}$ at step $(k+1)$ so we can denote it $\tau^{(k+1)}$. The IFA equation at step k implies:

$$\begin{aligned} & \gamma_2 * \tau^{(k+1)} = \gamma_1 * \tau^{(k)} \\ \Rightarrow \quad & \tau^{(k+1)} = \beta_{1,2} * \tau^{(k)} \end{aligned}$$

so that exploiting all equations from step 0 to step k gives:

$$\tau^{(k+1)} = \beta_{1,2}^{k+1} * \tau^{(0)}$$

To generate such unknown sequence τ we make use of an intermediate sequence θ of known values as described on Figure 11. At each step k, the value $\theta^{(k)}$ is defined such that $z_{1,1} = \tau^{(k)} = S(\theta^{(k)} \oplus k_{0,0})$ – and so such that $z_{1,2} = \tau^{(k+1)} = S(\theta^{(k+1)} \oplus k_{0,0})$ – in the following way:

- at step $k = 0$, the plaintext byte $m_{1+4\sigma_1}$ takes an arbitrary value and we have $\theta^{(0)} = m_{1+4\sigma_1} \oplus (\lambda_0 \oplus \lambda_{1+4\sigma_1})$,
- at step $k > 0$ the value of $\theta^{(k)}$ is defined as $\theta^{(k)} = m_{2+4\sigma_2} \oplus (\lambda_0 \oplus \lambda_{2+4\sigma_2})$ where $m_{2+4\sigma_2}$ is the value that produced the IFA at step $(k-1)$. The value of the plaintext byte ensuring that $z_{1,1} = S(\theta^{(k)} \oplus k_{0,0})$ at step k is given by $m_{1+4\sigma_1} = \theta^{(k)} \oplus (\lambda_0 \oplus \lambda_{1+4\sigma_1})$.

At some step $k = n-1$ we eventually observe that $\theta^{(n)} = \theta^{(0)}$ which implies:

$$\tau^{(n)} = \beta_{1,2}^{n} * \tau^{(0)} = \tau^{(0)}$$

from which we derive that the multiplicative order of $\beta_{1,2}$ is equal to n.

We can now change the roles of the indices to generate the sequences $\theta_{i,j}$ and $\tau_{i,j}$ related to the other $\beta_{i,j}$ (for $0 \leqslant i < j \leqslant 3$). Observe that it is always possible – and we

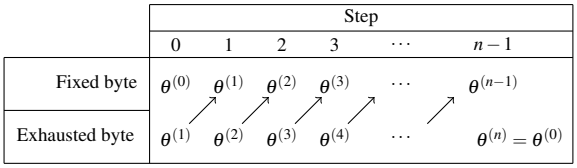

Fig. 11. Building the θ sequence based on a $\beta_{i,j}$ of order n

assume that the attacker proceeds this way – to choose adequately the plaintext bytes of the first step so that the six sequences $\theta_{i,j}$ share the same initial value $\theta_{i,j}^{(0)}$.

Having retrieved the orders of the six $\beta_{i,j}$, we can use them as constraints to reduce the number of possible quadruplets $(\gamma_0,\gamma_1,\gamma_2,\gamma_3)$.

Once we observe that the order of some $\beta_{i^\star,j^\star}$ is equal to 255, we can use the two corresponding sequences $\theta_{i^\star,j^\star}$ and $\tau_{i^\star,j^\star}$ as reference sequences which contain all non-zero values. From now on we will denote by θ and τ these reference sequences of length 255, and by $\beta = \beta_{i^\star,j^\star}$ the quotient that generates them. Remind that these sequences verify the two following properties:

$$\tau^{(k)} = S(\theta^{(k)} \oplus k_{0,0})$$

$$\tau^{(k)} = \beta^k * \tau^{(0)}$$

Remark 1. Retrieving the order of some $\beta_{i,j}$ is quite costly in terms of number of required faults. To obtain an IFA at step k we must exhaust from a set of $(256-k)$ values, so that this IFA requires an average of $(256-k)/2$ faults. Denoting $n_{i,j}$ the order of $\beta_{i,j}$, an average of $\Sigma_{k=0}^{n_{i,j}-1}(256-k)/2 \approx n_{i,j}(256-n_{i,j}/2))/2$ faults are needed to determine the sequence length. For example, about 2^{14} faults are required if $\beta_{i,j}$ is of full order.

As an optimization we remark that once the reference sequences θ and τ have been obtained, it is no more needed to generate the complete sequences for the other $\beta_{i,j}$. Indeed it suffices to obtain only the first IFA at step $k=0$ which gives the value $\theta_{i,j}^{(1)}$. One then identify $\theta_{i,j}^{(1)}$ as $\theta^{(x)}$ in the reference sequence and deduce that $n_{i,j} = 255/\gcd(255,x)$. As an example if more than one of the six orders are equal to 255, then only the determination of the first one costs 2^{14} faults, while the others only cost about 128 faults.

4.4 Reducing `MixColumns` Using Cross-Orders Relations

The orders $n_{i,j}$ of the $\beta_{i,j}$ all belong to the set $\{1,3,5,15,17,51,85,255\}$ of divisors of 255. Assume that for some pairs of indices (i_1,j_1) and (i_2,j_2) we have $n_{i_1,j_1} \mid n_{i_2,j_2}$ and $n_{i_1,j_1} > 1$ [4]. Then, due to the fact that they share their first element, the sequence θ_{i_1,j_1} is fully included in the sequence θ_{i_2,j_2}. We can thus identify the index t such that $\theta_{i_1,j_1}^{(1)} = \theta_{i_2,j_2}^{(t)}$ which implies that $\tau_{i_1,j_1}^{(1)} = \tau_{i_2,j_2}^{(t)}$ and thus gives the relation $\beta_{i_1,j_1} = \beta_{i_2,j_2}^t$.

All such cross-orders relations are inferred without the need of any additional fault, and can be used as constraints to further reduce the set of possible quadruplets $\{\gamma_0,\gamma_1,\gamma_2,\gamma_3\}$.

[4] In the case that $n_{i_1,j_1} = 1$, we already learn that $\gamma_{i_1} = \gamma_{j_1}$.

4.5 Reducing MixColumns by Using K_1 Relations

We still focus our IFA on the first S-Box of the second round and use relations obtained in Lemma 2 ($k_{1,i} \oplus k_{1,i+8} = \lambda_{i+4} \oplus \lambda_{i+8}$) to determine new constraints on MixColumns parameters. First we determine $mk_{i,1}$ and $mk_{i+8,2}$ as described in Corollary 1 in order to be able to choose plaintext bytes inducing $\gamma_1 * z_{1,1} = k_{1,i} \oplus S^{-1}(0)$ and $\gamma_2 * z_{1,2} = k_{1,i+8} \oplus S^{-1}(0)$. We also use knowledge of $mk_{0,0}$ already found to remove $k_{1,0}$ from the IFA equation. Then we exhaust the plaintext byte $m_{3+4\sigma_3}$ linked to $z_{1,3}$ until an IFA occurs. We can recognize $m_{3+4\sigma_3} \oplus (\lambda_0 \oplus \lambda_{3+4\sigma_3})$ as being some $\theta^{(p)}$, so we learn the index p such that $z_{1,3} = \tau^{(p)}$, from which we successively derive:

$$
\begin{aligned}
y_{2,i} &= 0 \\
\gamma_0 * z_{1,0} \oplus \gamma_1 * z_{1,1} \oplus \gamma_2 * z_{1,2} \oplus \gamma_3 * z_{1,3} \oplus k_{1,0} &= S^{-1}(0) \\
k_{1,0} \oplus S^{-1}(0) \ \oplus k_{1,i} \oplus S^{-1}(0) \ \oplus k_{1,i+8} \oplus S^{-1}(0) \ \oplus \gamma_3 * \tau^{(p)} \oplus k_{1,0} &= S^{-1}(0) \\
k_{1,i} \oplus k_{1,i+8} \oplus \gamma_3 * \beta^p * \tau^{(0)} &= 0 \\
\frac{\lambda_{i+4} \oplus \lambda_{1,i+8}}{\gamma_3 * \beta^p} &= \tau^{(0)}
\end{aligned}
$$

The unknown elements in the last equation are $\tau^{(0)}$ and the MixColumns parameters γ_3 and β. Two independent equations of this type can be combined to remove $\tau^{(0)}$ and create an equation involving only MixColumns parameters which can be used as an additional constraint to further reduce the set of possible MixColumns quadruplets.

Lemma 3. *At this point the informations acquired are sufficient to reduce the set of candidates for* $\{\gamma_0, \gamma_1, \gamma_2, \gamma_3\}$ *to* 255 *elements.*

Proof. Exploiting two K_1 relations from Lemma 2 gives us two equations that are:

$$
\left.
\begin{aligned}
\tau^{(0)} &= \frac{\lambda_{i+4} \oplus \lambda_{i+8}}{\gamma_3 * \beta^{p_1}} \\
\tau^{(0)} &= \frac{\lambda_{i+8} \oplus \lambda_{i+12}}{\gamma_3 * \beta^{p_2}}
\end{aligned}
\right\} \Rightarrow \ \beta^{p_1 - p_2} = \frac{\lambda_{i+4} \oplus \lambda_{i+8}}{\lambda_{i+8} \oplus \lambda_{i+12}}
$$

$$
\begin{aligned}
\Rightarrow \quad & (\frac{\gamma_{i^\star}}{\gamma_{j^\star}})^{p_1 - p_2} = \frac{\lambda_{i+4} \oplus \lambda_{i+8}}{\lambda_{i+8} \oplus \lambda_{i+12}} \\
\Rightarrow \quad & \gamma_{i^\star}^{p_1 - p_2} = \frac{\lambda_{i+4} \oplus \lambda_{i+8}}{\lambda_{i+8} \oplus \lambda_{i+12}} * \gamma_{j^\star}^{p_1 - p_2}
\end{aligned}
$$

As $\gamma_{i^\star}$ and $\gamma_{j^\star}$ are the only unknown values a choice for one of these two values will determine the second one. So it remains 255 valid pairs $(\gamma_{i^\star}, \gamma_{j^\star})$.

Previous step gave us cross-orders relations, in particular equations like:

$$\begin{aligned} & \beta_{i^\star,l}^{n_1} = \beta_{i^\star,j^\star}^{n_2} \\ \Rightarrow \quad & \left(\frac{\gamma_{i^\star}}{\gamma_l}\right)^{n_1} = \left(\frac{\gamma_{i^\star}}{\gamma_{j^\star}}\right)^{n_2} \\ \Rightarrow \quad & \gamma_l^{n_1} = \gamma_{j^\star}^{n_2} * \gamma_{i^\star}^{n_1-n_2} \end{aligned}$$

So for any valid pair $(\gamma_{i^\star}, \gamma_{j^\star})$ only one value for γ_l is valid, and we use the same method to uniquely identify the last parameter. Finally we obtain a reduced set of 255 valid candidates for `MixColumns` parameters.

Remark 2. For any candidate quadruplet the equations obtained in this step can be used to calculate the value of $\tau^{(0)}$, and thus the whole sequence $(\tau^{(k)})_{k=0,\ldots,254}$.

4.6 Retrieving `MixColumns` and `RotWord` Parameters

This step allows to recover the `MixColumns` and `RotWord` parameters and mostly uses previously acquired data. We use relations from the key schedule which involve the `RotWord` parameter η and we combine them with already obtained equations linking $k_{1,i}$ and $mk_{i,j}$ (for $i = 0,\ldots,3$ and for any j). We are able to recognize $mk_{i,j}$ as some $\theta^{(q)}$ value. We develop here the equations for $i = 0$, equations for others i can be found by the same way:

$$\begin{cases} k_{1,0} = k_{0,0} \oplus S(k_{0,12+\eta}) \oplus \rho^0 \\ k_{1,0} = \gamma_j * S(mk_{0,j} \oplus k_{0,0}) \oplus S^{-1}(0) \end{cases}$$

$$\begin{aligned} \Rightarrow \quad S(k_{0,12+\eta}) &= k_{0,0} \oplus S^{-1}(0) \oplus 1 \oplus \gamma_j * S(mk_{0,j} \oplus k_{0,0}) \\ &= \lambda_0 \oplus 1 \oplus \gamma_j * S(\theta^{(q_1)} \oplus k_{0,0}) \\ &= \lambda_0 \oplus 1 \oplus \gamma_j * \tau^{(q_1)} \end{aligned}$$

According to Remark 2, for any candidate for `MixColumns` parameters, the τ sequence is known so that the right part of the last equation can be computed and recognized as a value $\tau^{(q_2)}$ which is directly linked to a known value $\theta^{(q_2)}$:

$$\begin{aligned} \Rightarrow \quad & S(k_{0,12+\eta}) = \tau^{(q_2)} \\ \Rightarrow \quad & S(k_{0,12+\eta}) = S(\theta^{(q_2)} \oplus k_{0,0}) \\ \Rightarrow \quad & k_{0,12+\eta} = \theta^{(q_2)} \oplus k_{0,0} \\ \Rightarrow \quad & \theta^{(q_2)} = \lambda_0 \oplus \lambda_{12+\eta} \end{aligned}$$

The left part of last equation only depends on which of the 255 `MixColumns` candidates is assumed. The right part only depends on the `RotWord` parameter and can take 4 values. We have four equations (for $i = 0,\ldots,3$) but in most cases only two are sufficient. The use of one equation will let only 4 candidates for `MixColumns`, one per value of η. Then the probability is high that the intersection with candidates given by a second equation will let only one possible pair $((\gamma_0,\gamma_1,\gamma_2,\gamma_3),\eta)$.

4.7 Retrieving $S^{-1}(0)$

We are now able to calculate $k_{1,4}$:

$$\begin{cases} k_{1,0} = k_{0,0} \oplus \tau^{(q_2)} \oplus 1 \\ k_{1,4} = k_{1,0} \oplus k_{0,4} \end{cases}$$

$$\begin{aligned} \Rightarrow \qquad k_{1,4} &= k_{0,0} \oplus \tau^{(q_2)} \oplus 1 \oplus k_{0,4} \\ &= \tau^{(q_2)} \oplus 1 \oplus \lambda_0 \oplus S^{-1}(0) \ \oplus \lambda_4 \oplus S^{-1}(0) \\ &= \tau^{(q_2)} \oplus 1 \oplus \lambda_0 \oplus \lambda_4 \end{aligned}$$

We then use $k_{1,4}$ to derive $S^{-1}(0)$ from an equation obtained by Corollary 1:

$$\begin{aligned} k_{1,4} &= \gamma_j * S(mk_{4,j} \oplus k_{0,0}) \oplus S^{-1}(0) \\ \Rightarrow \qquad S^{-1}(0) \ &= \gamma_j * S(\theta^{(q_3)} \oplus k_{0,0}) \oplus k_{1,4} \\ &= \gamma_j * \tau^{(q_3)} \oplus k_{1,4} \end{aligned}$$

Once we have obtained $S^{-1}(0)$ we can deduce K_0 from the λ_i values. Knowing $k_{0,0}$ we infer the S-Box from all 255 equations $\tau^{(k)} = S(\theta^{(k)} \oplus k_{0,0})$. At that point we can compute the first round of the key schedule and derive K_1.

4.8 Retrieving `Rcon` Parameter

Since we know K_0, K_1 and all AES parameters except ρ, we can control the value of T_2. We are so able to exhaust values of $t_{2,0}$ to provoke an IFA at the output of the first S-Box of the third round. The IFA equation $y_{3,0} = 0$ allows to recover $k_{2,0} = t_{2,0} \oplus S^{-1}(0)$. Knowing $k_{2,0}$ we can simply calculate ρ as:

$$\begin{aligned} k_{2,0} &= k_{1,0} \oplus S(k_{1,12+\eta}) \oplus \rho^1 \\ \Rightarrow \qquad \rho &= k_{1,0} \oplus S(k_{1,12+\eta}) \oplus k_{2,0} \end{aligned}$$

4.9 Experimental Results

In order to estimate the number of faults necessary to fully recover the AES secret specifications we have performed PC-based simulations. For each of the successive steps we developed a program that simulates only that part of the attack so that we can evaluate the individual cost of each step. All these programs have been executed on a large number of simulation runs. Each run comprises the following features:

1. a secret AES-like block cipher is generated by drawing at random the set of its parameters complying with properties stated as in Section 2.2, as well as the extra property that at least one of the $\beta_{i,j}$ has order 255,
2. a secret key K is generated at random,
3. all parameters (or key knowledge) that are supposed to have been retrieved in previous steps are considered as known,

4. an oracle simulates a perfect IFA experiment: it takes as input all the AES parameters, the key, a plaintext and a fault position (expressed as a round and an S-Box index) and returns a Boolean value which indicates whether the output of this S-Box is zero (case of an IFA event) or not,
5. the attack step is performed by following the method described in the relevant section, and the number of calls to the oracle is counted.

Table 1. Experimental results of the FIRE attack on an unprotected implementation

Step	Number of faults
4.1 - Retrieving λ_i values	2 056
4.2 - Retrieving `ShiftRows`	138
4.3 - Retrieving $\beta_{i,j}$ orders	22 340
4.4 - Retrieving cross-orders relations	0
4.5 - Retrieving K_1 relations	916
4.6 - Retrieving `MixColumns` and `RotWord`	64
4.7 - Retrieving $S^{-1}(0)$	0
4.8 - Retrieving `Rcon`	128
Total	**25 642**

Table 1 gives the number of faults (IFA attempts) required by each step averaged over 10^6 runs. For sake of clarity we also mention the few steps that do not necessitate any fault while we have obviously not simulated them. As one can see the most costly step is the determination of the $\beta_{i,j}$ orders which requires about 22 300 faults even though we have implemented the optimization trick described in Remark 1. The other steps are much less costly so that about 25 600 faults on average are needed for recovering the complete specifications of a secret AES-like block cipher. While this figure may appear as a large number of faults, we emphasize that the entropy of recovered secret information in the case of a FIRE attack is usually much important compared to a classical key recovery. In the considered AES-like case, the total entropy to be retrieved amounts to about 1 734 bits[5] of information in addition to the 128 key bits.

4.10 Particular Cases

In this section we study two possible extended definitions of an AES-like cipher and describe how to adapt the FIRE attack presented above to these different cases. The first variant relaxes the choice of `MixColumns` parameters by allowing any invertible matrix of 16 coefficients instead of only circulant matrix. The entropy of the matrix is thus increased from about 32 to about 128 bits. The second variant relaxes the property

[5] Taking account all secret components: S-Box($\log_2(256!) \simeq 1684$ bits), `ShiftRows` (4 x 2 bits), `MixColumns` (4 x 8 bits), `RotWord` (2 bits), `Rcon` (8 bits).

that all Rcon[r] are defined as ρ^{r-1} which depend on the sole ρ byte value. Rather we allow independent ρ'_r bytes at each round $r = 1, \ldots, 10$. This increases from 8 to 80 bits the entropy of the Rcon parameter.

Full Entropy MixColumns Matrix The MixColumns matrix is not necessarily circulant and have 16 "independent" coefficients instead of only 4. The new MixColumns matrix is represented in Figure 12 where one can notice the new column-wise numbering of the coefficients.

$$\begin{bmatrix} \gamma_0 & \gamma_4 & \gamma_8 & \gamma_{12} \\ \gamma_1 & \gamma_5 & \gamma_9 & \gamma_{13} \\ \gamma_2 & \gamma_6 & \gamma_{10} & \gamma_{14} \\ \gamma_3 & \gamma_7 & \gamma_{11} & \gamma_{15} \end{bmatrix}$$

Fig. 12. Extended MixColumns matrix

Steps 4.1 and 4.2 are not impacted by this new setting and allow to retrieve λ_i values and ShiftRows parameters as previously described. Corollary 1 does not allow to obtain any value $mk_{i,j}$ any more but only for pairs of indices such that $i \equiv j \pmod 4$.

Corollary 2. *The knowledge of* $(\lambda_0, \ldots, \lambda_{15})$ *and* ShiftRows *parameters allows the attacker to retrieve any value* $mk_{i,j}$ *where* $i \equiv j \pmod 4$.

Proof. Let r and c denote respectively the row and the column of an element i. An IFA on value $y_{2,i}$ gives an equation involving the coefficients γ_{0+r}, γ_{4+r},γ_{8+r} and γ_{12+r} on a same row r:

$$\gamma_{0+r} * z_{1,4c} \oplus \gamma_{4+r} * z_{1,4c+1} \oplus \gamma_{8+r} * z_{1,4c+2} \oplus \gamma_{12+r} * z_{1,4c+3} \oplus k_{1,i} = S^{-1}(0)$$

As in Corollary 1, the attacker is able to force to zero three bytes of the considered input column, and exhaust the last one until an IFA occurs. Each position $4c+k$ (for $k = 0, \ldots, 3$) of the active byte induces one of the following equations:

$$\begin{aligned} \gamma_{0+r} * z_{1,4c} &= k_{1,i} \oplus S^{-1}(0) \\ \gamma_{4+r} * z_{1,4c+1} &= k_{1,i} \oplus S^{-1}(0) \\ \gamma_{8+r} * z_{1,4c+2} &= k_{1,i} \oplus S^{-1}(0) \\ \gamma_{12+r} * z_{1,4c+3} &= k_{1,i} \oplus S^{-1}(0) \end{aligned}$$

As one can see, $mk_{i,j}$ values can be obtained only for $j = 4k + r$ that is $j \equiv i \pmod 4$.

Due to Corollary 2, Step 4.3 has to be applied separately on each row r, allowing to recover six $\beta_{i,j}$ orders per row with $(i, j) \in \{0+r, 4+r, 8+r, 12+r\}$. We want to highlight that once the first $\beta_{i,j}$ order equal to 255 is discovered, the following orders (even from other rows) can be identified quickly according to optimization described

in Remark 1. As a consequence, the total cost of this step should not be significantly larger than for a circulant `MixColumns` matrix.

Step 4.4 derives cross-orders relations between two $\beta_{i,j}$ whenever the order of β_{i_1,j_1} divides that of β_{i_2,j_2}. Interestingly we notice that we can still derive such relations whether the two $\beta_{i,j}$ are issued from a same row or not. Indeed, due to Lemma 1 which allows to place a same value at output of any two arbitrary S-Boxes, we are able to start every sequence by a same $\theta^{(0)}$. Since all sequences are based on the same initial value we can identify the index t such that $\theta^{(1)}_{i_1,j_1} = \theta^{(t)}_{i_2,j_2}$, revealing the cross-orders relation.

In Section 4.5, we used two equations – out of the eight ones provided by Lemma 2 – to reduce to 255 the number of candidates for `MixColumns` parameters. The same method can be applied in the extended matrix case since we still have two equations available per each row. We thus use all eight equations – two equations per row – to reduce to 255 values each set of candidate for a row of `MixColumns`.

The method described in Section 4.6 have to be adapted and does not allow to fully retrieve `MixColumns` and `RotWord` parameters but creates a one-to-one relation between them. In our standard attack we use one of the four equations available that look like:

$$\begin{cases} k_{1,0} = k_{0,0} \oplus S(k_{0,12+\eta}) \oplus \rho^0 \\ k_{1,0} = \gamma_j * S(mk_{0,j} \oplus k_{0,0}) \oplus S^{-1}(0) \end{cases}$$

to reduce the number of `MixColumns` candidates to only four, exactly one per η candidate. This information is then combined with a second equation (e.g. one using $k_{1,1}$ and $mk_{1,j}$) to determine the correct couple of `MixColumns` and `RotWord` parameters. In the extended mode each equation involving $k_{1,i}$ is necessarily related to the row i of the `MixColumns` matrix. Thus we can exploit only one equation per row and reduce the `MixColumns` candidate set of each row to 4 elements, related to η candidates. We finally obtain a reduced set of only one `MixColumns` matrix for each `RotWord` candidate.

To end the attack notice that we have 2^2 candidates for the combination of `MixColumns` and `RotWord` parameters, 2^8 candidates for $S^{-1}(0)$ and 2^8 candidates for the `Rcon` parameter ρ. Each of the 2^{18} candidates on those elements allows to recover the S-Box table and then compute the entire key schedule. Indeed for each candidate the sequence τ is known since we know the `MixColumns` matrix, and $k_{0,0}$ is also known since we know $S^{-1}(0)$. We can therefore recover the S-Box by exploiting all 255 relations $\tau^{(k)} = S(\theta^{(k)} \oplus k_{0,0})$.

We are able to apply the method of Section 4.1 to the S-Boxes of last round and retrieve any arbitrary byte of K_{10}. For example when an IFA occurs on the i^{th} S-Box of last round we learn that $y_{10,i} = 0$ which implies that $k_{10,i'} = c_{10,i'}$ where $i' = (i - 4\sigma_{i \bmod 4}) \bmod 16$. We can then compute the key schedule under the 2^{18} candidates and determine the correct one by checking the predicted K_{10} against some of its actual bytes.

Remark 3. Compared to the circulant case, the fraction of matrices for which there exists at least one order of $\beta_{i,j}$ equals to 255 is increased from 95.28% to 99.99% due to the fact that it may appear on any row.

Extended `Rcon` Parameter Instead of defining $\texttt{Rcon}[r] = \rho^{r-1}$ parameters as being all dependent on the same byte ρ, we study in this section the case where they are all

independent. We denote them as ρ'_r (for $r = 1, \ldots, 10$) and the equation that defines the first byte $k_{r,0}$ of each round key becomes:

$$k_{r,0} = k_{r-1,0} \oplus S(k_{r-1,12+\eta}) \oplus \rho'_r$$

The `Rcon` parameters are only implied in equations at Steps 4.6, 4.7 and 4.8.

At Step 4.6 we retrieve `MixColumns` and `RotWord` parameters using two equations chosen amongst the four following available ones:

$$\begin{aligned}
k_{1,0} &= k_{0,0} \oplus S(k_{0,12+\eta}) \oplus \rho'_1 \\
k_{1,1} &= k_{0,1} \oplus S(k_{0,12+(1+\eta) \bmod 4}) \\
k_{1,2} &= k_{0,2} \oplus S(k_{0,12+(2+\eta) \bmod 4}) \\
k_{1,3} &= k_{0,3} \oplus S(k_{0,12+(3+\eta) \bmod 4})
\end{aligned}$$

As only one of them uses a `Rcon` parameter we can avoid it and apply this step without other modification.

At Step 4.7 we retrieve $S^{-1}(0)$ by calculating one of the four values $k_{1,4}$, $k_{1,5}$, $k_{1,6}$ and $k_{1,7}$. We took $k_{1,4}$ as example which is related to `Rcon` parameter by its relation to $k_{1,0}$:

$$\begin{cases}
k_{1,4} = k_{1,0} \oplus k_{0,4} \\
k_{1,0} = k_{0,0} \oplus S(k_{0,12+\eta}) \oplus \rho'_1
\end{cases}$$

As in the previous step we can avoid the difficulty by computing one of the three other values that are not linked to ρ'_1.

The last step detailed in Section 4.8 aims at recovering the only remaining unknown parameter ρ. At this step we argued that the control over state T_2 allows to calculate `Rcon` at round 2. This reasoning is applicable at any round because it only depends on the structure of the key schedule which is identical at each round.

With the extended `Rcon`, the only unknown AES parameters are the ten ρ'_r values. At first we have control over state T_1 that allows us to discover ρ'_1. Then we learn the value of K_1 which gives us the control over T_2 and we thus iteratively recover every values ρ'_r in the same way.

The extra-cost of extending the definition of `Rcon` parameter only resides in the fact that the method described in Section 4.8 have to be done for each round. The theoretical average cost of this step thus amounts 1275 faults instead of 127.5. This is a small increase compared to the total cost.

5 SCARE Attack in the Value Collisions Model

In this section we describe SCARE attacks performed in the value collisions model. We remind that in this model the attacker is able to detect when two S-Box access $y = S(x)$ and $y' = S(x')$ verify $x = x'$ and $y = y'$. We remind also that inter-traces setting means that the attacker is able to detect collisions between two S-Box in a same or in two different executions, and that intra-trace setting means that the attacker is able to detect collisions between two S-Box only in a same execution.

Section 5.1 describes an attack in this model when no countermeasure is implemented while Section 5.2 describes an attack in the presence of some classical countermeasures. In Section 5.3 we detail how to adapt the attack of Sect. 5.1 to a higher entropy `Rcon` parameter. Finally, provided that the attacker knows a priori the relative values of the key bytes, we extend in Sect. 5.4 the attack of Sect. 5.2 to a higher order masking (128-bit) protected implementation.

We remind the reader that the method detailed below was already published in [9], this version offers more effort of clarity.

5.1 SCARE Attack in the Value Collisions Model Without Countermeasures

In this section we describe how to recover the secret parameters of an AES implementation that does not feature any side-channel countermeasure. We proceed step by step, and the order of these steps has importance as each of them depends on the information retrieved in previous ones.

When this is relevant, we propose methods for both inter-traces and intra-trace settings.

Retrieving `ShiftRows` Parameters In the inter-traces setting we can easily recover the σ_i parameters. We first acquire a trace for a random plaintext, then we compare this trace with the four ones corresponding to a modification of a single plaintext byte $m_i\,(i = 0,\ldots,3)$. For each line i, observing which quadruplet of consecutive second round S-Boxes do not collide with the reference trace reveals the value of σ_i.

In the intra-trace scenario, things are a little more complex:

Lemma 4. *Assume a collision occurring between S-Box i in the first round, and S-Box j in the second round. There are two ways to destroy this collision by modification of a single plaintext byte: (i) either the active plaintext byte is at position i or, (ii) the active byte is any of the four bytes involved in the computation of $x_{2,j}$.*

Depending whether one of the four plaintext positions involved in the computation of $x_{2,j}$ is equal to i itself or not, there are respectively 4 or 5 active bytes that destroy the collision.

Definition 1. *We denote by `4-Collision` and `5-Collision` collisions that can be destroyed by 4 and 5 plaintext bytes respectively.*

Lemma 5. *The four bytes involved in the calculation of a same $x_{2,j}$ belong to different lines of the state matrix and are aligned on a same column after `ShiftRows` operation.*

To retrieve the `ShiftRows` parameters we first encrypt random plaintexts until a single collision occurs between a first round S-Box at position i and a second round S-Box at position j. Then for all $k \neq i$ we encrypt a modified plaintext where only m_k has changed, and identify whether 3 or 4 positions destroy the collision.

The first case (cf. red and $*$ in Figure 13) corresponds to a `4-Collision` and the three identified positions together with i are involved in the computation of $x_{2,j}$. The second case (cf. blue and $\boxtimes$ in Figure 13) corresponds to a `5-Collision` and the four

identified positions are related to $x_{2,j}$. In both cases, these four positions are equal to $\{4((c+\sigma_\ell) \bmod 4)+\ell\}_{\ell=0,\ldots,3}$ where $c=\lfloor j/4 \rfloor$ is the column of the collision. They are all different modulo 4 so that it is easy to infer the σ_ℓ parameters from them.

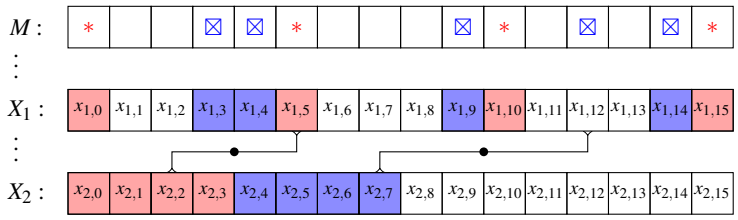

Fig. 13. Collision between $x_{1,5}$ and $x_{2,2}$ revealing a `4-Collision`(medium gray/red,$*$). And collision between $x_{1,12}$ and $x_{2,7}$ revealing a `5-Collision`(dark gray/blue,$\boxtimes$)

Retrieving K_0 and K_{10} up to a XOR with a Constant Byte The first step consists in detecting collisions between first round S-Boxes at two indices i and i' (cf. light gray/green boxes on Figure 14) on a same trace (or possibly on different ones in the inter-traces setting). Each such collision implies equality of two S-Boxes inputs and provides us with a linear relation between two key bytes:

$$\begin{aligned} x_{1,i}=x_{1,i'} &\Leftrightarrow (m_i \oplus k_{0,i})=(m_{i'} \oplus k_{0,i'}) \\ &\Leftrightarrow k_{0,i} \oplus k_{0,i'}=m_i \oplus m_{i'} \end{aligned}$$

Gathering several relations with random plaintexts eventually allows to relate all key bytes together. We now know each differential $\mu_{0,i,i'}=k_{0,i} \oplus k_{0,i'}$ and the key K_0 is thus retrieved up to a XOR with a constant byte. For example, it suffices to know the value of $k_{0,0}$ to compute other key bytes as $k_{0,i}=k_{0,0} \oplus \mu_{0,0,i}$.

Since we already retrieved the `ShiftRows` parameters we know which last round S-Box index any ciphertext byte is linked to. Similarly to the recovery of K_0, encrypting random plaintexts and observing collisions in the last round S-Boxes (cf. medium gray/red boxes on Figure 14) makes it possible to gather linear relations $\mu_{10,i,i'}=k_{10,i} \oplus k_{10,i'}=c_i \oplus c_{i'}$ which eventually reveal K_{10} up to a XOR with a constant byte. Note that the same set of traces can be used to recover both K_0 and K_{10} up to a constant.

Retrieving the S-Box Table A collision between a first round S-Box at index i and a last round S-Box at index j (cf. dark gray/blue boxes on Figure 14) implies that $x_{1,i}=x_{10,j}$ and $y_{1,i}=y_{10,j}$. Denoting $x=x_{1,i}$ and $y=y_{10,j}$, the collision reveals an S-Box relation $S(x)=y$ for two values $x=x' \oplus k_{0,0}$ and $y=y' \oplus k_{10,0}$ where $x'=m_i \oplus \mu_{0,0,i}$ and $y'=c_{j'} \oplus \mu_{10,0,j'}$ [6] are known from the attacker.

[6] Due to the `ShiftRows` the ciphertext byte related to the collision is located at index $j'=\ell+4((c-\sigma_c) \bmod 4)$ where $\ell=j \bmod 4$ and $c=\lfloor j/4 \rfloor$.

The S-Box table can thus be recovered by encrypting random plaintexts and observing such collisions (possibly on different traces) between first and last round S-Boxes. Once all 256 S-Box relations

$$S(x' \oplus k_{0,0}) = y' \oplus k_{10,0}$$

have been identified for all couples (x', y') the table S is recovered up to two XOR permutations on its inputs and outputs respectively.

When only collisions on the same trace are exploited, the relations are gathered like in the coupon collector problem. In that case we can save a large amount of traces by choosing the plaintexts so that all $x'_i = m_i \oplus \mu_{0,0,i}$ are different from each others and do not belong to already known relations.

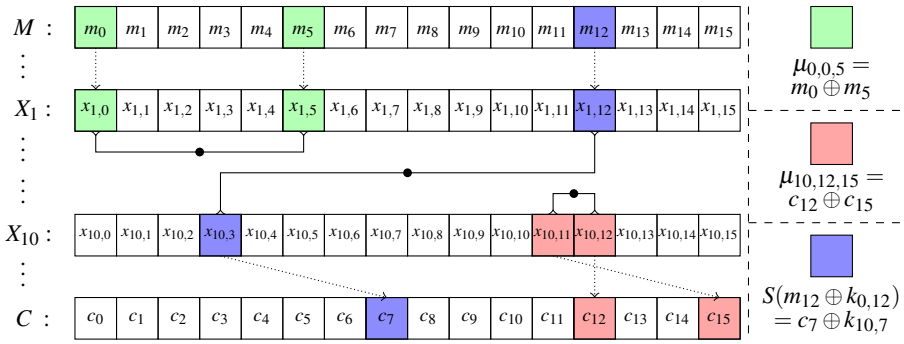

Fig. 14. Examples of collisions used in different attack steps in order to retrieve K_0 (light gray/green), K_{10} (medium gray/red) or the S-Box (dark gray/blue)

Retrieving K and all Key Schedule Parameters The next step is simply a 2^{26} offline exhaustive search that aims at recovering the absolute value of the key K as well as the key schedule parameters which are the amount of rotation η of the `RotWord` operation and the constant byte ρ that defines the `Rcon` vector.

For each candidate about $k_{0,0}$ and $k_{10,0}$ we know K_0, K_{10} and the S-Box table. It is then sufficient to make guesses also about the 2^2 values of η and the 2^8 values of ρ in order to be able to compute the key schedule and derive all round keys. Each of these 2^{26} candidates suggests a K_{10} value which is checked against the 128-bit known K_{10}. As this check on K_{10} is a 128-bit constraint, the probability of finding a false positive is overwhelmingly low, which has been confirmed by our simulations.

Note that a natural extension of the definition of `Rcon` would be to define each constant word as $\texttt{Rcon}[r] = (\rho_0^{r-1}, \rho_1^{r-1}, \rho_2^{r-1}, \rho_3^{r-1})$. Then the exhaustive search takes 2^{50} computations which may be considered as unaffordable. We propose in Section 5.3 an adaptation of our attack that can deal with such 32-bit entropy `Rcon` as well as with a full 320-bit entropy `Rcon` where all words are independent.

Retrieving the `MixColumns` Matrix At this point we have retrieved all secret parameters of the AES except the coefficients $\{\alpha_i\}_{i=0,\ldots,15}$ of the `MixColumns` matrix. We are so able to know the input of the first round `MixColumns` for each already acquired trace. As can be seen on Figure 15, each byte u_{i+4j} of the `MixColumns` output depends on 4 same-row parameters $\{\alpha_i, \alpha_{i+4}, \alpha_{i+8}, \alpha_{i+12}\}$:

$$u_{i+4j} = \alpha_i * v_{4j} \oplus \alpha_{i+4} * v_{4j+1} \oplus \alpha_{i+8} * v_{4j+2} \oplus \alpha_{i+12} * v_{4j+3}$$

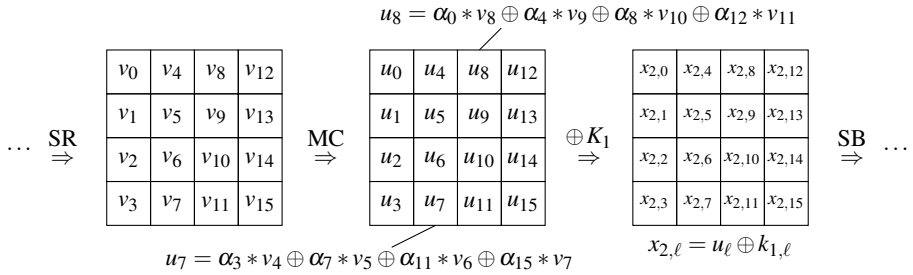

Fig. 15. Propagation of value v through `MixColumns` of first round

The goal is thus to obtain such equations by determining some input values $x_{2,\ell}$ of the second round S-Boxes, from which u_ℓ is inferred as $u_\ell = x_{2,\ell} \oplus k_{1,\ell}$ and the `MixColumns` input is derived from the plaintext. Gathering 4 independent equations involving the same set of parameters, and solving offline this system of equations, allows to recover a row of 4 parameters of `MixColumns` matrix. Finding 4 equations for each row allows to determine the whole matrix.

In both inter-traces and intra-trace settings no more traces are required for obtaining these equations as we can exploit traces already acquired for the previous steps. Amongst these traces we can find some $x_{2,\ell}$ values by noticing collisions occurring between $x_{2,\ell}$ and either some byte of known X_1 or some byte of known X_{10}. Our simulations demonstrate that the number of previously acquired traces always happens to be far from sufficient to get enough independent equations.

Experimental Results In order to verify the soundness of our attack and estimate the number of traces necessary to fully recover the AES secret specifications we have performed PC-based simulations. For each of the successive steps we developed a program that simulates only that part of the attack so that we can evaluate the individual cost of each step. All these programs have been executed on a large number of simulation runs that comprise the following features:

1. a secret AES-like block cipher is generated by drawing at random the set of its parameters complying with properties stated as in Section 2.2,
2. a secret key K is generated at random,

3. all parameters (or key knowledge) that are supposed to have been retrieved in previous steps are considered as known,
4. an oracle simulates a perfect collision detection: it takes as input all the AES parameters, the key, a plaintext and two S-Box positions (possibly at different rounds and/or on different traces) and returns a Boolean value which indicates whether the input/output pairs of these two S-Boxes are equal or not,
5. the attack step is performed by following the method described in the relevant section, and the number of traces used in the oracle queries is counted.

Table 2 presents the number of traces – averaged on 10000 runs – required by each step in both intra-trace and inter-traces settings. For sake of clarity we also mention the two last steps that do not necessitate any further trace.

Our attack on an unprotected implementation recovers the full set of secret AES parameters as well as the key within less than 400 traces on average by intra-trace analysis, and less than 100 traces when collisions between different traces can be exploited.

Table 2. Experimental results on an unprotected implementation

Step	# of traces		# of runs
	intra	inter	
Section 5.1 - Retrieving `ShiftRows`	11	5	10000
Section 5.1 - Reducing K_0 and K_{10} entropies to 8 bits	70	8	10000
Section 5.1 - Retrieving the S-Box	288	81	10000
Section 5.1 - Retrieving K and the key schedule	0	0	–
Section 5.1 - Retrieving `MixColumns`	0	0	10000
Total	**369**	**94**	

5.2 SCARE Attack in the Value Collisions Model With Countermeasures

In this section we consider a first-order side-channel protected implementation of the AES. Precisely we assume an implementation that jointly features two countermeasures:

The first countermeasure makes use of an 8-bit Boolean masking all along the data and the key schedule paths. Due to the considered attacker model, we are only concerned by the effect of this countermeasure on the inputs and outputs of the `SubBytes` operation. The masking of all other operations have no consequence on our attack. `SubBytes` uses a randomized version $\widetilde{S}$ of the S-Box table by means of two independent 8-bit input and output Boolean masks r_{in} and r_{out} such that $\widetilde{S}(x \oplus r_{in}) = S(x) \oplus r_{out}$ for all x. Due to memory and time constraints, typical embedded implementations of this countermeasures refresh the S-Box randomization only at each execution. We thus assume that the same randomized table is used for each input index and at each round of a same execution[7]. The main negative effect of this countermeasure for the attacker is

[7] A mask conversion is applied to masked intermediate values before or at the end of each round to adapt from the r_{out} of one round to the r_{in} of the next one.

that he is no more able to detect and exploit S-Box collisions from two different traces. Though, note that it is still possible to interpret collisions on $\widetilde{S}$ on a same trace as revealing collisions on the non-randomized S-Box. Thus, with this single countermeasure only, the intra-trace version of the attack described in Section 5.3 perfectly applies.

We also assume a second countermeasure which shuffles the 16 computations of $\widetilde{y}_i = \widetilde{S}(\widetilde{x}_i)$ at each round[8]. As a consequence, the observation of a collision between two (or more) computations of $\widetilde{S}(\widetilde{x})$ (possibly at different rounds) gives no information about the index of the $\widetilde{x} = x \oplus r_{in}$ input bytes. The attacker is thus limited to observe the number of different S-Box inputs at each round, and how many occurrences of each of them there are. To capture this limited attacker capacity, we introduce the following definition:

Definition 2. *Let's define an n-structure (or more simply a structure) of type* $n_1^{(t_1)} n_2^{(t_2)} \dots n_s^{(t_s)}$ *of elements of E the set of all n-tuples of elements of E (with* $n = \Sigma_k t_k n_k$*) such that* $t = \Sigma_k t_k$ *distinct elements appear in the tuple with* $n_1, n_2, \dots, n_s$ *occurrences of each of them respectively.*

For example, any X_r made of all distinct S-Box input bytes belongs to a structure of type $1^{(16)}$ of elements of $GF(2^8)$. As another example, the 16-tuple:

$$X_1 = (13, 47, 173, 47, 86, 119, 13, 47, 119, 223, 205, 119, 37, 88, 200, 5)$$

exhibits a $1^{(8)}2^{(1)}3^{(2)}$ structure as 13 appears twice, and 47 and 119 appear three times each.

Retrieving K_0 up to a XOR with a Constant Byte The first step consists in executing the AES with random plaintexts until finding one such that the first round `SubBytes` presents a unique n-fold colliding value (a $1^{(16-n)}n^{(1)}$ structure). Only about a couple of traces are needed on average to find such a reference trace. Then, for each $i = 0, \dots, 15$, one modifies m_i and observes whether the collision disappears (or its multiplicity n is reduced). The set I of indices for which this happens verifies:

$$\forall\, i, i' \in I,\ k_{0,i} \oplus k_{0,i'} = m_i \oplus m_{i'}$$

where m_i and $m_{i'}$ are the byte values from the reference plaintext.

By comparing the reference trace with at most 16 modified ones, one should result with $|I| = n$ in most cases. However, if $n = 2$, a non-detection may occur when the change of an input byte involved in the initial collision makes it collide with another not initially colliding one. In such rare case, it should be sufficient to change again the different message bytes to reveal all which of them are involved in the initial collision.

Once a set of n colliding indices is identified, n different key bytes are linearly related together. Repeating this process to exploit different reference plaintexts exhibiting $1^{(16-n)}n^{(1)}$ structures, eventually allows to relate all key bytes to each others. K_0 is then retrieved up to a XOR with a constant byte (e.g. $k_{0,0}$).

[8] Here also the shuffling of other AES operations such as `ShiftRows`, `MixColumns`, `AddRoundKey`, etc. have no influence on the attack proposed in the considered S-Box collisions model.

Notice that two tricks allow to reduce the number of traces required to relate all key bytes together. First, when a linear relation is known for all key bytes at indices belonging to some subset J, one should choose the reference plaintext such that X_1 bytes belonging to J are all different. The second trick is an early abort of the process of determining the set I for a reference plaintext: as soon as a new relation is found that involves a key byte from J, one can skip considering other indices from J .

Retrieving K_{10} up to a XOR with a Constant Byte If the attacker can query a decipher oracle with chosen ciphertexts, it is possible to recover the value of K_{10} up to a XOR with a constant by a similar method than that used to recover K_0. One first encrypts random plaintexts until finding one such that the last round `SubBytes` presents a $1^{(16-n)}n^{(1)}$ structure. Let C be the ciphertext. For modified ciphertexts C', differing from C by only one byte c_i, one then encrypts $M' = \mathrm{AES}_K^{-1}(C')$ and observes whether the collision in the last round S-Boxes disappeared. By the same principle as for K_0, it is thus possible to identify relations like $k_{10,i} \oplus k_{10,i'} = c_i \oplus c_{i'}$. Accumulating sufficiently many such relations eventually reveals K_{10} up to a XOR with a constant, with the same complexity than for K_0.

When the attacker does not have access to a decipher oracle, it is still possible to recover K_{10}. For random plaintexts, we exploit only traces which show no collision in the last round S-Boxes (a $1^{(16)}$ structure). In that case we know that for each index pair (i,i'), $\mu_{10,i,i'}$ is not equal to $c_i \oplus c_{i'}$. Starting from lists of all possible values for all $\mu_{10,i,i'}$, and accumulating such negative information, we end up with sufficiently many lists containing only one remaining value so that K_{10} is finally recovered up to a XOR with a constant byte (e.g. $k_{10,0}$).

Note that we can do better by exploiting (possibly a posteriori) traces with collisions as well. Whenever some $\mu_{10,i,i'}$ is known one can detect when a collision occurs between S-Boxes related to bytes c_i and $c_{i'}$. If it happens that this identified collision on X_{10} is the only one on this trace, then one can infer negative information as above for all index pairs (j,j') not involved in the collision.

Retrieving the S-Box Table At this point, we know K_0 and K_{10}, each up to a XOR with a constant. We now show how to recover the S-Box table for each candidate about these two constants. We are seeking couples $(x' = x \oplus k_{0,0}, y' = y \oplus k_{10,0})$ verifying $S(x) = y$ as in Section 5.1. To that end we select wisely chosen plaintexts such that X_1 contains five different byte values $x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$ and $x^{(6)}$. As depicted on Figure 16 each value is repeated a different number of times, so that when a collision occurs the colliding value can be identified based on the collision order.

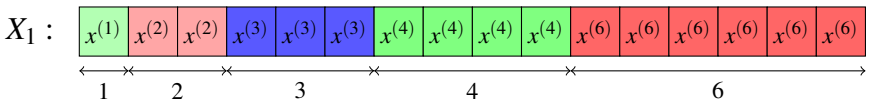

Fig. 16. Example of state X_1 having five values with different numbers of occurrences

If a collision occurs between one (or several) of the five input values $x^{(i)}$ and some X_{10} byte then at least 240 values y' are invalidated for pairing with $x' = x^{(i)} \oplus k_{0,0}$. For any other non-colliding $x^{(j)}$, one can invalidate up to 16 values y' for pairing with $x' = x^{(j)} \oplus k_{0,0}$. After each execution one should propagate the negative information as much as possible. For example if it is known that x' is necessarily paired with y' then one can invalidate all couples (x', y'') and (x'', y') with $x'' \neq x'$ and $y'' \neq y'$. This, in turn can reveal another assured pair, and so on.

The same set of $x^{(i)}$ can be used multiple times if needed by just changing their positions. Nevertheless, as a wise strategy for choosing the $x^{(i)}$ values, we suggest to select values $x' = x^{(i)} \oplus k_{0,0}$ with the least number of invalidated y' values. The rational behind this criterion is to maximize the expected gained information.

Retrieving K and all Key Schedule Parameters The same offline 2^{26} exhaustive search (not impacted by the countermeasures) as in Section 5.1 can be conducted to retrieve K and the key schedule parameters.

Retrieving the `MixColumns` Matrix and the `ShiftRows` Parameters Knowing K and the S-Box we are able to fully control vectors X_1 and Y_1. Let's encrypt the plaintext M_0 defined by $m_i = k_{0,i} \oplus S^{-1}(0)$ for all i, so that $Y_1 = (0, \ldots, 0)$. For that reference plaintext the output of the `MixColumns` is also all zeroes and we have $X_2 = K_1$. By analysing the trace of this execution (or simply because we know K_1) we obtain the number n_0 of second round S-Box inputs which collide with the value $S^{-1}(0)$ of X_1 in the first round (cf. Figure 17).

Without loss of generality, assume that we want to recover the first column $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ of the `MixColumns` matrix. Let's modify only one message byte m_{4c} and denote by $v = y_{1,4c}$ the value of the active cell at S-Box output. The column number $((c - \sigma_0) \bmod 4)$ takes value $(v, 0, 0, 0)$ at input, and $(\alpha_0 v, \alpha_1 v, \alpha_2 v, \alpha_3 v)$ at output of the `MixColumns`, whereas all other columns remain unchanged. Assuming that the active quadruplet of bytes of X_2 does not contain the value $S^{-1}(0)$ in the reference execution[9], by exhausting v we can identify four values[10] v_0, v_1, v_2 and v_3 which induce more than n_0 occurrences of $S^{-1}(0)$ in the second round S-Box inputs (cf. Figure 18). Since each of these values has provoked an extra $S^{-1}(0)$ in X_2, we know that for some unknown permutation π of $\{0, 1, 2, 3\}$ the following holds:

$$\forall i = 0, \ldots, 3 \qquad \alpha_i v_{\pi(i)} \oplus k_{1,i+4((c-\sigma_0) \bmod 4)} = S^{-1}(0)$$

For each possible σ_0 this system of equations suggests a set of 24 values (one per candidate about π) for the targeted column of coefficients $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$.

[9] The opposite case should be rare and is easily detectable by observing a reduction of the number of occurrences of $S^{-1}(0)$. In that case, simply modify c to change the column of the active cell.

[10] It may happen that less than four values are identified when one or two of them produce multiple extra $S^{-1}(0)$ values. In such case, these special v values should be counted as many times as their collision order.

At this point, we can change the value of c and repeat this process with the active cell at the top of another column. For each possible σ_0 we obtain a new set of 24 candidates for the column of coefficients. By intersecting the two sets, with high probability, there remains only one column value for the correct σ_0 and none for the incorrect ones, which therefore reveals $(\alpha_0, \alpha_1, \alpha_2, \alpha_3)$ and σ_0.

We can do better by having two active bytes on the same row. This allows the attacker to gather the eight v values that produce extra collisions with only one stone. The number of permutation candidates to exhaust becomes 8! (instead of $2 \times 4!$) which is still affordable.

By repeating this attack with the active cell located on the other rows, we can successively recover the three other columns of coefficients together with the corresponding `ShiftRows` parameters.

In the rare cases where it remains an indecision about some column(s) of `MixColumns` coefficients (and possibly `ShiftRows` parameters), it can be solved by checking the few AES-like candidates against a known plaintext/ciphertext pair.

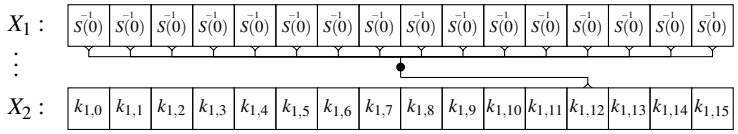

Fig. 17. Collision revealing that $n_0 = 1$

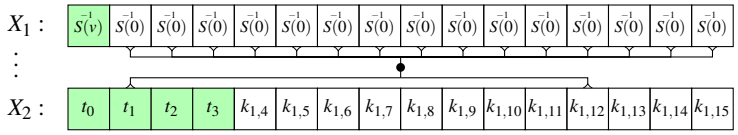

Fig. 18. Presence of an extra collision ($n = 2$) gives exploitable information

Experimental Results We performed simulations of the different attack steps described in this section in a similar way as for the unprotected implementation case described in Section 5.1.

Table 3 presents the number of traces – averaged on 10000 runs – required by each step of the attack. For sake of clarity we also mention the offline exhaustive search of Section 5.2 that does not necessitate any further trace.

Overall, less than 4000 traces are required to fully recover to whole set of secret AES parameters in the more realistic scenario of a classical first-order protected implementation and no prior information about the key.

We emphasize that the entropy of secret information in the case of a SCARE attack is usually much important compared to a classical key recovery. In the considered AES-

like case, the total entropy to be retrieved amounts to about 1 830 bits[11] of information in addition to the 128 key bits.

Table 3. Experimental results on a masked and shuffled implementation

Step	# of traces	# of runs
Section 5.2 - Reducing K_0 entropy to 8 bits	246	10 000
Section 5.2 - Reducing K_{10} entropy to 8 bits	1 408	10 000
Section 5.2 - Retrieving the S-Box	1 263	10 000
Section 5.2 - Retrieving K and the key schedule	0	–
Section 5.2 - Retrieving MC and SR	910	10 000
Total	**3 827**	

5.3 Extension of `Rcon` Parameter in Case of Unprotected Implementations

Our attack against unprotected implementations described in Section 5.1 assumes an 8-bit entropy `Rcon` vector made of 10 words $\texttt{Rcon}[r] = (\rho^{r-1}, 0, 0, 0)$. Two natural extensions would be to define a 32-bit entropy or a full 320-bit entropy `Rcon` vectors of constant words respectively equal to $(\rho_0^{r-1}, \rho_1^{r-1}, \rho_2^{r-1}, \rho_3^{r-1})$ and $(\rho_{r,0}, \rho_{r,1}, \rho_{r,2}, \rho_{r,3})$ for $r = 1, \ldots, 10$. In the first case the exhaustive search of Section 5.1 necessitates 2^{50} computations which is a quite intensive task, while it is completely infeasible in the second case. In this section we present an adaptation of this attack step that exploits new collisions to recover `Rcon` parameters in both extended cases. We only detail here the 320-bit entropy case, the method can trivially be adapted to the 32-bit entropy case.

We remind that the `Rcon` parameters are not involved in first attack steps, this induce that we are able to follow previous methods to recover K_0 up to $k_{0,0}$, K_{10} up to $k_{10,0}$, `ShiftRows` parameters and the S-Box table up to $(k_{0,0}, k_{10,0})$. On the contrary the offline 2^{26} exhaustive search of Section 5.1 now becomes a non-affordable 2^{338} exhaustive search.

Inter-Traces Version This new version of the attack step is split into three phases. The first phase uses 2^8 traces to determine a list of 2^{16} K_1 candidates. The second phase exploits key schedule constraints to find the correct K_1 and $\rho_{1,i}$ parameters. Finally the third phase allows to retrieve every other $\rho_{r,i}$ parameters.

In the first phase we use the knowledge of $\mu_{0,i,j}$ values to encrypt plaintexts such that all 16 bytes of the first round S-Boxes are identical. As a consequence, one of these 2^8 encryptions verifies that Y_1 contains only zeroes but we do not known which one since we do not know $k_{10,0}$. For this particular execution the output of `MixColumns` is also an all zeroes state so that the input state X_2 of the second round S-Boxes is equal to K_1.

[11] Taking account of all secret components: S-Box($\log_2(256!) \simeq 1684$ bits), `ShiftRows` (4 x 2 bits), `MixColumns` (16 x 8 bits), `RotWord` (2 bits), `Rcon` (8 bits).

As we are in inter-traces setting, for each one of the 2^8 traces we are able to identify the value of X_2 up to $k_{0,0}$. This is done by finding a collision between each $x_{2,j}$ on this trace and some $x_{1,i}$ on another one. We then have: $x_{2,j} = x_{1,i} = m_i \oplus k_{0,i} = (m_i \oplus \mu_{0,0,i}) \oplus k_{0,0}$. To sum up we have identified 2^{16} candidates for K_1, one per candidate for the couple $(k_{0,0}, k_{10,0})$.

The second offline phase uses the key schedule structure to obtain K_1. For each candidate about $(k_{0,0}, k_{10,0})$, each candidate for η and $\rho_{1,0}$ allows to compute $k_{1,0}$, $k_{1,4}$, $k_{1,8}$ and $k_{1,12}$. So for each $(k_{0,0}, k_{10,0})$ there are 2^{10} candidates for those 4 bytes of K_1 which can be checked against the corresponding K_1 candidate found in first phase for this $(k_{0,0}, k_{10,0})$ pair. With high probability there will be only one matching configuration which reveals the correct values of $k_{0,0}$, $k_{10,0}$, η, $\rho_{1,0}$ and K_1. At this point, the attacker knows K_0, K_{10}, the S-Box table, η , K_1 and $\rho_{1,0}$. Knowing K_0, η, and K_1, he can easily calculate $\rho_{1,1}$, $\rho_{1,2}$ and $\rho_{1,3}$ but also apply the method of Section 5.1 to recover `MixColumns` parameters.

The third phase uses already acquired traces in order to find missing $\rho_{r,i}$ parameters. First, we aim to recover second round $\{\rho_{2,0}, \rho_{2,1}, \rho_{2,2}, \rho_{2,3}\}$. For that purpose we just need to find K_2. Indeed these $\rho_{2,i}$ parameters are involved in the derivation of K_2 from K_1, so if K_1 and K_2 are known the attacker is able to calculate corresponding `Rcon` parameters.

As K_0, S-Box table, `ShiftRows` and `MixColumns` parameters and K_1 are known, for every available trace we are able to know the input T_2 of the second round `AddRoundKey`. As we are able to determine every $x_{3,i}$ by collisions, we easily infer $K_2 = T_2 \oplus X_3$. Once K_2 is recovered we use the key schedule relations to obtain $\{\rho_{2,0}, \rho_{2,1}, \rho_{2,2}, \rho_{2,3}\}$.

Similarly, the knowledge of K_2 allows to calculate T_3 state, then identifying X_4 by collisions reveals K_3 from which $\{\rho_{3,0}, \rho_{3,1}, \rho_{3,2}, \rho_{3,3}\}$ are derived, and so on for every round. At the last round, no collisions are necessary to find K_{10} which is directly computed as $Z_{10} \oplus C$.

Intra-Trace Version In the case where only intra-trace collisions are available, attacker have to adapt the previous phases.

In the first phase, for each 2^8 encryptions described above, we obtain values of $x_{2,i}$ by observing collisions with modified plaintexts. First we keep constant those specific plaintext bytes so that $x_{2,0}$ to $x_{2,7}$ remain unchanged. By modifying other plaintext bytes we give ourselves opportunities to observe collisions between a first round S-Box input and every $x_{2,i}$ for $i = 0, \ldots, 7$. Then we can change the roles of fixed and free plaintext bytes to keep $x_{2,8}$ to $x_{2,15}$ constant and recover these eight remaining bytes of X_2.

The second phase is performed offline so it can be done as in the inter-traces setting.

The third phase needs to find collisions between $x_{3,i}$ and a known value in order to determine $x_{3,i}$ and infer $k_{2,i}$. At this point X_1, X_2 and X_{10} are known so that there are 48 bytes that may collide with every $x_{3,i}$. For any trace and any $x_{3,i}$ the probability that a collision exists between $x_{3,i}$ and one of these bytes is $p = 1 - (1/256)^{48} \approx 0.17$. We have sufficiently many available traces to recover every $x_{3,i}$ – not necessarily on the same trace – from which we infer all bytes of K_2. For successive next rounds things are even easier since at each round we have 16 more known bytes (X_r) that may collide with each $x_{r+1,i}$. Remark that not all bytes of K_r are needed to recover the $\rho_{r,i}$ parameters

since only one byte per line is actually sufficient. For example if we know $k_{2,8}$ we can successively compute $k_{2,4} = k_{2,8} \oplus k_{1,8}$, $k_{2,0} = k_{2,4} \oplus k_{1,4}$ and $\rho_{r,0} = k_{2,0} \oplus k_{1,0} \oplus S(k_{1,12+\eta})$.

5.4 Extention to Higher Order Masking

We consider a higher order masking scheme using 16 masked `SubBytes` tables, one per S-Box in a round. We still combine it with the random order countermeasure. Note that the attack on the high-order masking scheme that we propose in this section assumes that the attacker already knows the relative values $\mu_{0,i,j}$ of the key bytes.

With these countermeasures we can only detect collisions within a same trace and between two S-Boxes at a same position in different rounds. These collisions are rarer but give the extra information that they are vertically aligned. To identify that a collision between two S-Boxes at two different rounds is actually vertically aligned, we need to reject false positives by encrypting the same plaintext twice or more. Since all input and output masks are independent, the probability that S-Box i at some given round collide with S-Box $i' \neq i$ at some other given round is about 2^{-16} since both input values and output values must collide. As there are 240 couples (i, i') the probability of a false positive between the S-Boxes of these two rounds is about 2^{-8}. Thus, if the collision still exists after multiple encryptions – two should be sufficient in most cases – the probability is high that it is a true collision between aligned values masked with the same input and output bytes. In the rest of the Section 5.4 when we say that a collision occurs between two rounds, that means that it has been confirmed to be an aligned collision.

Retrieving `ShiftRows` Parameters We encrypt random plaintexts until a collision occurs between first and second rounds. We then encrypt the same plaintext with only one byte modified each time and we check if the collision stays or disappears. We learn whether it is a `4-Collision` or a `5-Collision` by counting the number of positions that make the collision disappear.

In the case of a `4-Collision` we learn that the four active bytes are on a same column after `ShiftRows` operation but we do not known which column. We can thus infer relative σ_i values – say its shape – up to an additive constant modulo 4.

Note that in the case of a `4-Collision` we also have the extra information that at least one of the σ_i is zero. This is because the collision is aligned and the byte position that is involved in the collision also belongs to the column implied in the computation of the second round colliding byte.

In the case of a `5-Collision` we have three lines containing only one active element from which we can infer the differences between the σ_i related to these lines. The last line contains two active elements. One of them, which is implied in the computation of the second round colliding byte, is aligned with the actives bytes of the three other lines. The other is aligned with the second round colliding byte and does not belong to the active column. We can not distinguish between these two cases but for each two hypothesis on the position of the collision the active column is determined and we can derive the absolute values of the σ_i parameters.

The procedure to precisely identify the `ShiftRows` parameters is to generate several first/second round collisions until obtaining either two 5-`Collisions` or one 4-`Collision` and one 5-`Collision`.

If we obtain two 5-`Collisions` we identify the correct solution by intersecting the two pairs of solutions given by each collision.

If we obtain one 4-`Collision` and one 5-`Collision` we can use the shape information given by the 4-`Collision` to decide between the two solutions given by the 5-`Collision`.

Note that in the case where all σ_i are zeroes only 4-`Collisions` can occur. This particular case is easily identified from the first collision since all σ_i are the same (knowledge of the shape) and one of them is zero, which implies that they are all zeroes.

Retrieving K_{10} up to a XOR with a Constant Byte We target first/last rounds collisions but the shuffling countermeasure prevents us to determine the precise position of the collision. We circumvent this problem by generating a first/second/last rounds triple collision from an existing first/second rounds collision. To that end, once we obtain a first/second rounds collision at a known position, we maintain it by fixing the 4 or 5 plaintext bytes that destroy it, and we change the other bytes at will until noticing a triple collision with the last round also.

From a triple collision we infer that the first and last rounds S-Boxes have same inputs and same outputs. We thus have $S(m_i \oplus k_{0,i}) = c_j \oplus k_{10,j}$ where i is the known position of the collision, j is deduced from i and the `ShiftRows` parameters, and only $k_{0,i}$ and $k_{10,j}$ are unknown.

In order to recover K_{10} up to a constant byte we use two equations obtained by such triple collisions. A first triple collision at position i gives us a couple (m_i, c_j) verifying $S(m_i \oplus k_{0,i}) = c_j \oplus k_{10,j}$. A second one at an arbitrary position $i' \neq i$ provides an other couple $(m_{i'}, c_{j'})$ verifying the same kind of equation.

The trick to obtain a suitable second triple collision is to give to m'_i the value $m_i \oplus \mu_{0,i,i'}$ (so that $x'_i = x_i$) and change bytes that are implied in the computation of $x_{2,i'}$ until we get a first/second rounds collision. Then we keep this collision by fixing the concerned plaintext bytes, and change the other free bytes until we obtain the expected triple collision. As $x'_i = x_i$, we derive that $c_j \oplus k_{10,j} = c_{j'} \oplus k_{10,j'}$ which reveals $\mu_{10,j,j'} = c_j \oplus c_{j'}$.

Retrieving S-Box Table up to Two Bytes In the previous attack step we showed how to provoke triple collisions that give equations like $S(m_i \oplus k_{0,i}) = c_j \oplus k_{10,j}$. Each such equation reveals a couple (x,y), where $x = m_i \oplus \mu_{0,0,i}$ and $y = c_j \oplus \mu_{10,0,j}$, verifying $S(x \oplus k_{0,0}) = y \oplus k_{10,0}$. Accumulating 255 independent equations reveals the S-Box table up to $(k_{0,0}, k_{10,0})$.

Retrieving K and all Key Schedule Parameters This part of the attack is performed offline and can be done as described in Sect. 5.1. After this step the only remaining unknown parameters are those of `MixColumns` matrix.

Retrieving `MixColumns` Coefficients In order to recover α values we use the traces where triple collisions where identified in previous steps. We know the 4 or 5 first round S-Box outputs involved in the collision and the value of the second round colliding byte. Those values are related together by an equation in four unknown α values. Accumulating four independent such equations allows to solve the system and recover a line of four α parameters.

If necessary we can generate new triple collisions with selected positions in order to recover remaining undetermined α values if any.

6 SCARE Attack in the Hamming Weight Collisions Model

We remind that in the Hamming weight collisions model the attacker is able to detect when two S-Box calls $y = S(x)$ and $y' = S(x')$ simultaneously verify $HW(x) = HW(x')$ and $HW(y) = HW(y')$. In this section we describe how to recover the secret parameters of an AES implementation that does not feature any side-channel countermeasure. We proceed step by step, and the order of these steps has importance as each of them depends on the information retrieved in previous ones. At the end of this section we consider how we can also deal with implementations featuring a first-order masking countermeasure.

We remind the reader that the method detailed below is the novel contribution of the paper. It stands in the intra-trace setting.

6.1 Definitions

Definition 3. *For $0 \leqslant a, b \leqslant 8$, we define the sets $\Psi_{a,b}$ and $\Omega_{a,b}$ as follows:*

$$\Psi_{a,b} = \{x \,|\, \mathrm{HW}(x) = a, \mathrm{HW}(S(x)) = b\}$$
$$\Omega_{a,b} = \{y \,|\, \mathrm{HW}(S^{-1}(y)) = a, \mathrm{HW}(y) = b\}$$

Any two byte values belonging to the same $\Psi_{a,b}$ (respectively the same $\Omega_{a,b}$) set are inputs (respectively outputs) of two S-Box that will collide under the Hamming weight collisions model defined in Section 3.2. One can remark that $|\Psi_{a,b}| = |\Omega_{a,b}|$ since S is a permutation. Figure 19 presents the repartitions of cardinals of Ψ sets for the standard S-Box table and the number of occurrences of each of them.

In some steps of the following attack we use particular sets which verify $|\Psi_{a,b}| = 1$. Due to uniqueness of value in the set, a collision implying two values from such a set reveals that inputs and outputs of both implied S-Boxes are actually equal, instead of only sharing the same Hamming weights. For instance there are 12 such sets in the case of the standard AES.

Lemma 6. *There are at least two Ψ sets such that $|\Psi_{a,b}| = 1$*

Proof. As 0 (respectively 255) is the only element to have a Hamming weight equals to 0 (respectively 8) we have $|\Psi_{0,\mathrm{HW}(S(0))}| = |\Omega_{8,\mathrm{HW}(S(255))}| = 1$

In some following steps we have to exhaust S-Box inputs in order to provoke a collision, but in that configuration we are able to reduce the number of values to test. We use the fact that all elements of a Ψ set will have the same behaviour in term of collision/non-collision, then we have to encrypt only one value per Ψ set.

$\lvert\Psi_{a,b}\rvert$	b 0	1	2	3	4	5	6	7	8	
0	0	0	0	0	1	0	0	0	0	1
1	0	0	2	0	1	3	2	0	0	8
2	0	2	3	8	5	4	4	2	0	28
3	1	1	4	17	16	10	5	2	0	56
a 4	0	3	9	11	21	16	9	1	0	70
5	0	1	7	10	19	14	3	2	0	56
6	0	0	3	7	5	8	4	0	1	28
7	0	1	0	2	2	1	1	1	0	8
8	0	0	0	1	0	0	0	0	0	1
	1	8	28	56	70	56	28	8	1	

$\sum_{a=0}^{8} \lvert\Psi_{a,b}\rvert$ (column sums)

$\sum_{b=0}^{8} \lvert\Psi_{a,b}\rvert$ (row sums)

$\lvert\Psi_{a,b}\rvert$	#
0	34
1	12
2	8
3	5
4	4
5	3
7	2
8	2

$\lvert\Psi_{a,b}\rvert$	#
9	2
10	2
11	1
14	1
16	2
17	1
19	1
21	1

Fig. 19. $\lvert\Psi_{a,b}\rvert$ repartition (at left) and occurrences (at right) for the standard AES S-Box

6.2 Considerations about Ψ and Ω sets

Before describing the attack we start by some considerations about Ψ and Ω sets and explain how information is obtained about them, and how they are used during the attack steps.

At some point we need to have identified Ψ sets. By this we mean that we have partitioned the set of all 256 S-Box inputs into the sets $\Psi_{a,b}$ that respectively contain all inputs which collide together. Note that for each identified $\Psi_{a,b}$ we know the input Hamming weight a but possibly do not know the Hamming weight b of their images through S.

Symmetrically we also need to have identified Ω sets with the meaning of a partition of all S-Box outputs into the $\Omega_{a,b}$ sets that respectively contain all outputs which collide together. In this case we know the output Hamming weight b but possibly do not know the Hamming weight a of their pre-images through S.

Note that there exists a one-to-one mapping between the set of all $\Psi_{a,b}$ and the set of all $\Omega_{a,b}$. We will also have to identify this mapping. For each $\Psi_{a,b}$ (whose b was unknown) we need to identify its $\Omega_{a,b}$ companion set (whose a was unknown). Obviously, since two related Ψ and Ω sets share the same a and b, once such a link is identified these a and b parameters are then known. Notice that all y from an $\Omega_{a,b}$ are the images through S of all x from its $\Psi_{a,b}$ companion, but we are possibly not able to link individual values together yet.

Figure 20 depicts the three types of information that we exploit from every acquired trace. Provided that we know K_0, a first round collision allows to put together two S-Box inputs in a same $\Psi_{a,b}$. For example in the case of the light gray/green collision we put $x_{1,0}$ and $x_{1,5}$ in a same $\Psi_{a,b}$ where $a = \mathrm{HW}(x_{1,0})$ and b is a priori unknown. Similarly, with knowledge of K_{10} and the `ShiftRows` parameters, we can put $y_{10,11}$ and $y_{10,12}$ in

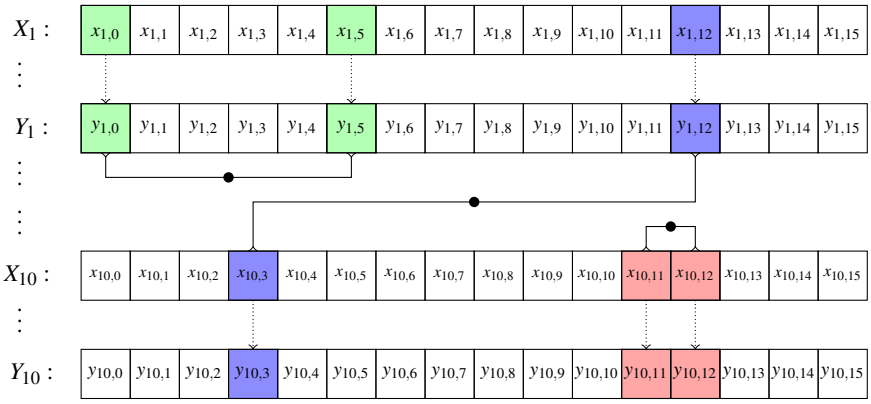

Fig. 20. Examples of collisions revealing Ψ (light gray/green), Ω (medium gray/red) or links (dark gray/blue) information

a same $\Omega_{a,b}$ – a unknown and $b = \mathrm{HW}(y_{10,11})$ – once we observe the medium grey/red last round collision. If we know all K_0, K_{10} and the `ShiftRows` parameters, then we can induce from a first/last round collision, such as the dark grey/blue one, that the Ψ set containing $x_{1,12}$ is linked to the Ω set containing $y_{10,3}$, even if we have not identified all elements of these sets yet.

During the attack every collision/non-collisions of those three types are detected and the related information is memorized. Indeed, any (i) first round non-collision, (ii) last round non-collision or (iii) first/last round non-collision is informative. They respectively inform that (i) the two x inputs do not belong to the same Ψ set, (ii) the two y outputs do not belong to the same Ω set, (iii) the first round input x and the last round output y respectively belong to Ψ and Ω sets that are not linked to each other.

These three types of information are useful during the attack. Indeed we can infer from them that:

- if a particular value is in a set of cardinal 1, then any collision involving this element is a collision of values and not only a collision of Hamming weights,
- if two sets $\Psi_{a,b}$ and $\Omega_{a,b}$ are known to be linked together, and if we identify an S-Box input (respectively output) to be in $\Psi_{a,b}$ (respectively $\Omega_{a,b}$), then the corresponding S-Box output (respectively input) is known to belong to a reduced list of candidates, namely the elements of $\Omega_{a,b}$ (respectively $\Psi_{a,b}$).

The prerequisite to start building Ψ sets is the knowledge of K_0 because we have to know $x_{1,i}$ values involved in a first round collision. We can proceed even if we know K_0 up to its opposite – uncertainty between K_0 and $\overline{K_0}$ – because in this case we also know the $x_{1,i}$ values up to their opposite. In that case we can choose an arbitrary candidate for K_0 and start to build Ψ sets under this assumption. If a later information eventually reveals that we have chosen the wrong candidate we simply have to transform every $\Psi_{a,b}$ set already build to its actual opposite $\Psi_{8-a,b}$ which contains the opposites of each of its elements. For sake of clarity we will not mention in the sequel this potential permutation which does not perturb any of our attack steps.

Similarly, the prerequisites to start building Ω sets is the knowledge of K_{10} and of the `ShiftRows` parameters, and those to start identifying links is the knowledge of all K_0, K_{10} and `ShiftRows` parameters. Notice that the same reasoning as above – related to the uncertainty between a key and its opposite – also holds for the building of the Ω sets (if we only know K_{10} up to its opposite) and for the linking of Ψ and Ω sets.

6.3 Retrieving K_0 up to 2^{16} Candidates

In the first step we aim to find every $k_{0,i}$ up to its opposite leading to 2^{16} candidates for K_0.

Consider two different traces where a first round collision occurs on one trace at positions i and j for plaintext bytes m_i and m_j, and another first round collision occurs on the other trace, also at positions i and j, for plaintexts bytes $m'_i \neq m_i$ and $m'_j = m_j$. Then have $x_j = x'_j$ from which we infer that $\mathrm{HW}(x_i) = \mathrm{HW}(x'_i)$. Given m_i and m'_i we can invalidate all candidates $k_{0,i}$ that do not verify $\mathrm{HW}(m_i \oplus k_{0,i}) = \mathrm{HW}(m'_i \oplus k_{0,i})$.

We accumulate several constraints for each key byte until we reduce to only two opposite candidates for every $k_{0,i}$. Indeed $k_{0,i}$ and $\overline{k_{0,i}}$ are necessarily undistinguishable as:

$$\begin{aligned}
\mathrm{HW}(m_i \oplus k_{0,i}) = \mathrm{HW}(m'_i \oplus k_{0,i}) &\Leftrightarrow 8 - \mathrm{HW}(m_i \oplus k_{0,i}) = 8 - \mathrm{HW}(m'_i \oplus k_{0,i}) \\
&\Leftrightarrow \mathrm{HW}(m_i \oplus k_{0,i} \oplus 255) = \mathrm{HW}(m'_i \oplus k_{0,i} \oplus 255) \\
&\Leftrightarrow \mathrm{HW}(m_i \oplus \overline{k_{0,i}}) = \mathrm{HW}(m'_i \oplus \overline{k_{0,i}})
\end{aligned}$$

In order to provoke those types of collisions we fix one half of the plaintext to arbitrary values while the second half is randomly chosen. Doing this we increase the probability that two traces exhibit a suitable pair of collisions with i and j respectively belonging to the free part and to the fixed part. When all key bytes at free positions have been recovered, we proceed again in the same way after having changed the roles of the fixed and free parts.

6.4 Retrieving K_0 up to Two Candidates

In this step we determine K_0 up to its opposite with only one new trace. As each $k_{0,i}$ is known up to its opposite we are able to encrypt a chosen plaintext with $m_i \in \left\{k_{0,i}, \overline{k_{0,i}}\right\}$ inducing that $m_i \oplus k_{0,i} = x_{1,i} \in \{0, 255\}$. The positions of collisions allow to identify two sets of indices, one where $x_{1,i} = 0$ and the other where $x_{1,i} = 255$ while we do not know which is which. By simply complementing the plaintext bytes at one of these two sets of indices we obtain a 16-byte vector which is equal either to K_0 or $\overline{K_0}$, from which we derive all $\mu_{0,i,j}$ values.

The knowledge of K_0 up to its opposite allows us to start building Ψ sets as detailed in Section 6.2.

6.5 Retrieving `ShiftRows` Parameters

In this step we search for an already acquired trace that exhibits a first/second round collision where we can identify the first round S-Box input as one of the few particular

values that we know to belong to a $\Psi_{a,b}$ set of cardinal 1. This implies that the two colliding values are equals. If we can not find this configuration in already acquired trace we manage to generate one.

At this point we are in a suitable situation to retrieve `ShiftRows` parameters by the method of Sect. 5.1 based on the analysis of `4-Collisions` and `5-Collisions`.

6.6 Retrieving K_{10} up to 2^{16} Candidates

Knowing `ShiftRows` parameters allows to relate ciphertext bytes positions with positions of colliding S-Boxes in the last round. We use the same method used for K_0 bytes in Section 6.3 but applied on last round S-Box outputs. We exploit pairs of traces that exhibit a collision between $y_{10,i}$ and $y_{10,j}$ on one trace and a collision between $y'_{10,i}$ and the same $y_{10,j}$ on the other one. This way we can progressively recover each $k_{10,i}$ byte up to its opposite.

The main difference with K_0 method is that we can not fix part of the outputs of last round S-Boxes, but this is counterbalanced by the fact that we are looking for last/last round but also for first/last rounds collisions. Indeed a first round S-Box input $x_{1,j}$ can play the same role as $y_{10,j}$. We use already acquired traces and this method almost never needs new traces.

6.7 Retrieving K_{10} up to Two Candidates

We know every $k_{10,i}$ up to its opposite. Contrarily to K_0 we do not need any new encryption to retrieve K_{10} up to an inversion because first/last rounds collisions give us enough information. Consider two traces (possibly the same) which both exhibit a first/last rounds collision, and such that the two first round S-Box inputs involved in the collisions have been identified as belonging to the same Ψ set. This implies that the two last round S-Box outputs have the same Hamming weight and thus $\mathrm{HW}(c_i \oplus k_{10,i}) = \mathrm{HW}(c_j \oplus k_{10,j})$ where c_i and c_j are the ciphertext bytes related to both collisions. If this Hamming weight is different from 4 we can differentiate whether a candidate on the pair $(k_{10,i}, k_{10,j})$ is either: (i) both non-inverted or both inverted, or (ii) one inverted and the other not:

$$(i)\begin{cases}\mathrm{HW}(c_i \oplus k_{10,i}) = \mathrm{HW}(c_j \oplus k_{10,j}) \\ \mathrm{HW}(c_i \oplus \overline{k_{10,i}}) = \mathrm{HW}(c_j \oplus \overline{k_{10,j}})\end{cases} \qquad (ii)\begin{cases}\mathrm{HW}(c_i \oplus k_{10,i}) = 8 - \mathrm{HW}(c_j \oplus \overline{k_{10,j}}) \\ \mathrm{HW}(c_i \oplus \overline{k_{10,i}}) = 8 - \mathrm{HW}(c_j \oplus k_{10,j})\end{cases}$$

The knowledge of K_{10} up to its opposite and `ShiftRows` parameters allows us to start building Ω sets as detailed in Section 6.2.

6.8 Identifying Ψ Sets, Ω Sets and the Ψ-Ω Links

Further steps need that we complete the identification of Ψ sets, Ω sets and links relating one Ψ set and its Ω set companion. We start by extracting the maximum information from the already acquired traces by the three types of collisions described in Section 6.2. Then we have to generate new traces optimized to reduce as fast as possible the missing information.

First we complete the identification of the Ψ sets by encrypting plaintexts with bytes chosen in order to maximize the number of couples $(x_{1,i}, x_{1,j})$ containing two values for which it is not yet determined if they collide or not.

Then we choose plaintexts bytes in order to maximize the number of $x_{1,i}$ values from not already linked Ψ sets until we complete Ω sets and Ψ-Ω links information. We use as much as possible the relations between sets to speed up the information recovery.

6.9 Retrieving K_1

In order to retrieve `MixColumns` and `SubBytes` parameters in the two following steps we need the value of K_1.

In first round the `Rcon` constant is always $\rho^0 = 1$. This induces that K_1 only depends on K_0, η, $S(k_{0,12})$, $S(k_{0,13})$, $S(k_{0,14})$ and $S(k_{0,15})$:

$$\begin{aligned} k_{1,0} &= S(k_{0,12+(0+\eta) \bmod 4}) \oplus k_{0,0} \oplus \rho^0 & \qquad k_{1,2} &= S(k_{0,12+(2+\eta) \bmod 4}) \oplus k_{0,2} \\ k_{1,1} &= S(k_{0,12+(1+\eta) \bmod 4}) \oplus k_{0,1} & \qquad k_{1,3} &= S(k_{0,12+(3+\eta) \bmod 4}) \oplus k_{0,3} \end{aligned}$$

$$k_{1,i} = k_{1,i-4} \oplus k_{0,i} \text{ for all } i = 4, \ldots, 15$$

In a first offline phase we compute a list of possible candidates for $(k_{1,0}, k_{1,1}, k_{1,2}, k_{1,3})$. Note that knowing K_0, K_1 only depends on these four bytes. For each 16 candidates for (K_0, K_{10}, η) we identify the Ψ sets that respectively contain $k_{0,12}$, $k_{0,13}$, $k_{0,14}$ and $k_{0,15}$. Their Ω companion sets are respective candidates lists for all S-Box outputs involved in above equations. We can thus build a list of $(k_{1,0}, k_{1,1}, k_{1,2}, k_{1,3})$ candidates for each guess on (K_0, K_{10}, η).

In a second phase we use the knowledge acquired about Ψ and Ω sets. For each two candidates on K_{10} we know $S^{-1}(0)$, so for each two candidates on K_0 we can encrypt plaintexts that induce 0 values as output of any first round S-Box. If we choose to force to zero a whole column at input of `MixColumns` then on this column the second round S-Box input bytes have same values as K_1.

A first/second rounds collision allows to know the Ψ set that contains the concerned K_1 byte. We thus reduce the candidates list for one of the $\{k_{1,0}, k_{1,1}, k_{1,2}, k_{1,3}\}$ bytes. Indeed, knowing K_0, any list of candidates for $k_{1,i}$ can be transformed in a list of candidates for $k_{1,i \bmod 4}$.

This reduction process leads to only one K_1 value in 96.7% of cases. When several K_1 are possible[12] we split the attack in several branches: for each possible K_1 we proceed with the next steps of the attack and eventually find an inconsistency that invalidates this value. If several K_1 remain valid up to the end of the attack we then have to test each resulting AES-like specification against a plaintext/ciphertext pair.

Beside K_1 this step also recovers K_0, K_{10}, ρ and up to four pairs (x, y) verifying $S(x) = y$.

[12] In these cases there are only 2.1 candidates on average.

6.10 Retrieving α Values

We aim to recover the 16 α_i coefficients of the `MixColumns` matrix. To this purpose we use 4 times a method allowing to recover 4 of them.

The knowledge of K_0 and K_{10} allows identify all couples (x,y) verifying $S(x) = y$ where x and y are in Ψ and Ω sets that contain only one element, and in particular $(S^{-1}(0),0)$. We can thus encrypt plaintexts such that a so-called V-vector $(V,0,0,0)$ is a column input of the matrix multiplication, with V a non zero known byte. This V-vector gives an output column that expresses as $(\alpha_0 * V, \alpha_1 * V, \alpha_2 * V, \alpha_3 * V)$. This induces that the corresponding second round S-Boxes will have input values $x_{2,i}$:

$$x_{2,i} = (\alpha_j * V) \oplus k_{1,i} \;\Rightarrow\; \alpha_j = \frac{x_{2,i} \oplus k_{1,i}}{V}$$

where only α_j and $x_{2,i}$ are unknown. A first/second rounds collision can determine the $\Psi_{a,b}$ set that contains $x_{2,i}$, so we obtain $|\Psi_{a,b}|$ candidates for $x_{2,i}$, and therefore for α_j. By applying this method with other V we can intersect the set of candidates for every α_j until they contain only one element.

The position of V value in the V-vector relates to which α_i subset we target. If V is in position c then we target the column c of coefficients: $(\alpha_{4*c+0}, \alpha_{4*c+1}, \alpha_{4*c+2}, \alpha_{4*c+3})$.

To speed up the α_i recovery, we choose to place two V-vectors in two input columns, and use the 8 remaining free bytes to exhaust values in order to provoke the expected collisions.

6.11 Retrieving the S-Box Table

With the knowledge of α coefficients we are able to complete the S-Box table. We search in available traces (or create new ones if needed) cases where only one of the four input bytes of a first round `MixColumns` column is unknown. For example we have (W,V_1,V_2,V_3) with only W unknown. Then $x_{2,i} = \alpha_0 * W \oplus \alpha_4 * V_1 \oplus \alpha_8 * V_2 \oplus \alpha_{12} * V_3 \oplus k_{1,i}$ where only W and $x_{2,i}$ are unknown. We use the same method than in previous step: a collision between $x_{2,i}$ and a first round byte gives the Ψ set this value belongs to, and so a list of candidates for W.

One can remark that we are in better conditions than for α values because for every value x we already know in which Ω set is $y = S(x)$, so we can start to intersect when the first collision is found instead of waiting for a second collision. Also we can take as y candidates from Ω only those that have not been already associated with an other x value.

An other remark is that when it remains only one non-associated element in a Ψ set its S-Box image is necessarily the last available value in the corresponding Ω set. We use this property to gain extra information during the analysis of already acquired traces and to wisely choose our plaintexts in case we have to generate new traces.

Once a pre-image is found for some W we can check anew available traces to find cases where W was present with a single other unknown value W' in an input column: (W',W,V_1,V_2). That configuration was not exploitable because there were two unknown bytes but as we just found W we can now use this trace to obtain information about W'.

This step does need new traces only in very rare cases.

6.12 Retrieving ρ

The only parameter which remains unknown is the `Rcon` constant ρ. In order to find this value we run an offline version of the key schedule on K_0 for each ρ candidate. Only the correct ρ value produces the known K_{10} as output of the key schedule.

6.13 Experimental Results

We followed the same process of simulations as for other attacks. The only difference stays in the oracle that gives Hamming weight collisions detection instead of value collisions.

Table 4 presents the number of traces – averaged on 10000 runs – required by each step. For sake of clarity we also mention the steps that do not necessitate any further trace.

Our attack on an unprotected implementation recovers the full set of secret AES parameters as well as the key within about 250 traces on average in the intra-trace setting. A fortiori significantly less traces would be needed in the inter-traces setting.

Table 4. Experimental results on an unprotected implementation in the Hamming weight collisions model

Step	# of traces	# of runs
Section 6.3 - Reducing K_0 entropy to 16 bits	151.9	10000
Section 6.4 - Reducing K_0 entropy to 1 bit	1.	10000
Section 6.5 - Retrieving `ShiftRows`	11.3	10000
Section 6.6 - Reducing K_{10} entropy to 16 bits	0.007	10000
Section 6.7 - Reducing K_{10} entropy to 1 bit	0.	10000
Section 6.8 - Identifying Ψ sets, Ω sets and Ψ-Ω links	53.8	10000
Section 6.9 - Retrieving K_1	12.3	10000
Section 6.10 - Retrieving `MixColumns`	26.2	10000
Section 6.11 - Retrieving `SubBytes`	0.2	10000
Section 6.12 - Retrieving `Rcon`	0.	10000
Total	**256.6**	

6.14 SCARE Attack in the Hamming Weight Collisions Model with a Masking Countermeasure

In previous Sections 6.3 to 6.12 we described a SCARE attack in the intra-trace setting that applies on unprotected implementations in the Hamming weight collisions model

In the case of an implementation protected by the same first-order Boolean masking countermeasure as considered in Sect. 5.2, the attack is no more feasible due to the effect of the input and output masks that spoil the detection of Hamming weight collisions. Indeed false collisions may appear when $\mathrm{HW}(x \oplus r_{in}) = \mathrm{HW}(x' \oplus r_{in})$ and $\mathrm{HW}(y \oplus r_{out}) = \mathrm{HW}(y' \oplus r_{out})$ while either $\mathrm{HW}(x) \neq \mathrm{HW}(x')$ or $\mathrm{HW}(y) \neq \mathrm{HW}(y')$.

Also true Hamming weight collisions may be undetected when either $\mathrm{HW}(x \oplus r_{in}) \neq \mathrm{HW}(x' \oplus r_{in})$ or $\mathrm{HW}(y \oplus r_{out}) \neq \mathrm{HW}(y' \oplus r_{out})$ while $\mathrm{HW}(x) = \mathrm{HW}(x')$ and $\mathrm{HW}(y) = \mathrm{HW}(y')$.

A means to circumvent these false decision cases is to encrypt multiple times the same plaintext and detect those collisions which persist over these encryptions. As expected this procedure removes the false positive cases, but as a side effect it also removes cases of Hamming weight collisions which are not value collisions.Actually, only collisions of values ($x = x'$ and $y = y'$) remain detectable over multiple maskings with different mask pairs.

The Hamming weight collision oracle thus becomes a value collision oracle through multiple encryption of masked executions. We can take advantage of that and apply the attack of Sect. 5.2. In this case, the number of traces required for the attack is obviously multiplied by the number of times each plaintext is encrypted. As a conservative option repeating each plaintext 5 times should be enough to exclude virtually all cases of false positive and lead to an attack still feasible while requiring about 20 thousands traces[13].

We did not assumed above that the order of operations was shuffled at each execution. In that case it is more tricky to turn the Hamming weight collision oracle into a value collisions oracle. We let the study of this case as an open problem.

Note also that since we only assume the Boolean masking countermeasure, when applying the attack of Sect. 5.2 we could certainly simplify it in many ways since the information about collisions positions is available. Another remark is that the attack of Sect. 5.4 – or probably a more simple version – could also apply in the case of a high-order 128-bit masking. Here also we let the studies of these ideas to future research work.

7 Security Recommendations Related to our Attacks

All attacks described in this paper assume a software implementation of an AES-like block cipher on an 8-bit microprocessor.

First remark that an efficient way to thwart the IFA based FIRE attack of Sect. 4 is to implement the first-order Boolean masking counter-measure. In this case an IFA event reveals that the value $y' = y \oplus r_{out}$ read from memory on the faulted execution was zero but this may happen whatever the value of y depending on the mask value. Thus no information about the relevant value y is obtained in this case.

Let's consider the SCARE attacks. In the realistic Hamming weight collisions model we show in Sect. 6 that a SCARE attack is possible even in the presence of first-order masking. In Sect. 5.2 we also describe a SCARE attack that jeopardizes implementations with joint first-order masking and shuffling countermeasures in the value collisions model. A (probably costly) extension of these attacks to the case of a high-order masking is also presented in Sect. 5.4 in the case where the attacker has prior knowledge about the key. Our findings tend to limit the confidence on the strength of these kinds of countermeasures against SCARE attacks in both collisions models. This is the reason

[13] Actually this figure is certainly overestimated as in many cases there may be no collisions at all amongst the rounds we are interested in. This situation can often be identified with only one or two encryptions.

why we think that an efficient means to make our attacks infeasible would be to mitigate the side-channel signal in order to make the detection of collisions impossible or at least quite difficult. This can be achieved by inserting time randomization or activating current noise generation possibly featured by the chip. Obviously for better security these last countermeasures should preferably be implemented together with masking and shuffling.

Note also that all our attacks should hardly apply on hardware implementations of the AES, particularly if the underlying hardware features dual-rail logic that strongly decreases the side-channel leakage and provides an opportunity to detect stuck-at faults.

Finally as our attacks stand in the chosen plaintext scenario, any application whose specification do not allow a free choice of the AES input would be invulnerable to our attacks.

8 Conclusion

In this paper we have investigated the problem of retrieving the secret parameters of an AES-like block cipher derived from the standard AES function by modifying part of or all its constant parameters (S-Box table, `ShiftRows` rotations, `MixColumns` matrix coefficients as well as `RotWord` rotation and `Rcon` constant).

Our reverse-engineering study considered several kinds of physical attacks on software 8-bit implementations of the secret function: an ineffective fault analysis (IFA) based on a stuck-at-zero fault model where the fault is injected on the output of an S-Box, and several side-channel based collision power analysis under different collisions models where the collision is detected between the input/output pairs of two different S-Boxes.

We have described in detail four different attacks and provided precise PC-based simulations results for each of them:

(a) An IFA-based FIRE attack that applies on an unprotected implementation. This attack requires about 25 thousands fault attempts.
(b) A SCARE attack under the value collisions model on an unprotected implementation. Depending on whether the attacker is able to detect collisions from different traces or not, the proposed attack requires about one hundred or four hundreds traces respectively.
(c) A SCARE attack under the value collisions model on an implementation jointly protected by a first-order Boolean masking and a random shuffling of operations. We showed that such a secured implementation can be attacked by analysing collisions on less than four thousands traces.
(d) A SCARE attack under the more realistic Hamming weight collisions model on an unprotected implementation. This attack is particularly efficient and only requires analysing about 250 traces.

We also proposed – without having simulated them – several extensions of our attacks: three extensions of attacks (a) and (b) to alternative definitions of `Rcon` constant with a larger entropy, and an extension of attack (c) to the case of a high-order Boolean masking when the attacker has some prior knowledge about the key. We also argued

that multiple encryption of the same plaintext makes attack (c) also applicable under the relaxed Hamming weight collisions model.

Beside providing different security recommendations with respect to our attacks, we also identified several problems that could be further studied as future research works.

Our results demonstrates the necessity to protect the implementation of a block cipher against physical attacks even though its specifications are not public.

Acknowledgements

This work has been conducted under the framework of the MARSHAL+ (Mechanisms Against Reverse-engineering for Secure Hardware and Algorithms) research project, subsidized by FUI 12, and co-sponsored by the competitiveness clusters System@tic and SCS.

Practical results presented in this paper have been partly performed on the CALI computing cluster of university of Limoges, funded by the Limousin region, XLIM, IPAM and GEIST institutes, as well as the university of Limoges.

Bibliography

[1] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In Burton S. Kaliski, Jr, editor, *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 513–525. Springer-Verlag, 1997.

[2] Alex Biryukov, Andrey Bogdanov, Dmitry Khovratovich, and Timo Kasper. Collision Attacks on AES-Based MAC: Alpha-MAC. In Paillier and Verbauwhede [27], pages 166–180.

[3] Andrey Bogdanov. Improved Side-Channel Collision Attacks on AES. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography – SAC '07*, volume 4876 of *Lecture Notes in Computer Science*, pages 84–95. Springer, 2007.

[4] Andrey Bogdanov. Multiple-Differential Side-Channel Collision Attacks on AES. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES '08*, volume 5154 of *Lecture Notes in Computer Science*, pages 30–44. Springer, 2008.

[5] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51. Springer-Verlag, 1997.

[6] Éric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In Joye and Quisquater [15], pages 16–29.

[7] Christophe Clavier. Secret External Encodings Do not Prevent Transient Fault Analysis. In Paillier and Verbauwhede [27], pages 181–194.

[8] Christophe Clavier. An Improved SCARE Cryptanalysis Against a Secret A3/A8 GSM Algorithm. In Patrick Drew McDaniel and Shyam K. Gupta, editors, *International Conference on Information Systems Security – ICISS '07*, volume 4812 of *Lecture Notes in Computer Science*, pages 143–155. Springer, 2007.

[9] Christophe Clavier and Antoine Wurcker. Reverse Engineering of a Secret AES-like Cipher by Ineffective Fault Analysis. In Wieland Fischer and Jörn-Marc Schmidt, editors, *Fault Diagnosis and Tolerance in Cryptography – FDTC '13*, pages 119–128. IEEE Computer Society Press, 2013.

[10] Christophe Clavier, Benedikt Gierlichs, and Ingrid Verbauwhede. Fault Analysis Study of IDEA. In Tal Malkin, editor, *Topics in Cryptology – CT-RSA '08*, volume 4964 of *Lecture Notes in Computer Science*, pages 274–287. Springer, 2008.

[11] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Improved Collision-Correlation Power Analysis on First Order Protected AES. In Preneel and Takagi [28], pages 49–62.

[12] Christophe Clavier, Quentin Isorez, and Antoine Wurcker. Complete SCARE of AES-like Block Ciphers by Chosen Plaintext Collision Power Analysis. In Goutam Paul and Serge Vaudenay, editors, *International Conference on Cryptology in India – INDOCRYPT '13*, Lecture Notes in Computer Science, pages 116–135. Springer, 2013.

[13] Rémy Daudigny, Hervé Ledig, Frédéric Muller, and Frédéric Valette. SCARE of the DES. In John Ioannidis, Angelos D. Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security – ACNS '05*, volume 3531 of *Lecture Notes in Computer Science*, pages 393–406. Springer-Verlag, 2003.

[14] Sylvain Guilley, Laurent Sauvage, Julien Micolod, Denis Réal, and Frédéric Valette. Defeating Any Secret Cryptography with SCARE Attacks. In Michel Abdalla and Paulo S. L. M. Barreto, editors, *Progress in Cryptology – LATINCRYPT '10*, volume 6212 of *Lecture Notes in Computer Science*, pages 273–293. Springer, 2010.

[15] Marc Joye and Jean-Jacques Quisquater, editors. *Cryptographic Hardware and Embedded Systems – CHES '04: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, 2004. Springer-Verlag.

[16] Marc Joye, Jean-Jacques Quisquater, Sung-Ming Yen, and Moti Yung. Observability Analysis – Detecting When Improved Cryptosystems Fail. In Bart Preneel, editor, *Topics in Cryptology – CT-RSA '02*, volume 2271 of *Lecture Notes in Computer Science*, pages 17–29. Springer-Verlag, 2002.

[17] Çetin Kaya Koç and Christof Paar, editors. *Cryptographic Hardware and Embedded Systems – CHES '00, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, 2000. Springer-Verlag.

[18] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.

[19] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, 1999.

[20] Rita Mayer-Sommer. Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards. In Koç and Paar [17], pages 78–92.

[21] Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In Koç and Paar [17], pages 238–251.

[22] Thomas S. Messerges, Ezzy A. Dabbish, and Robert H. Sloan. Investigations of Power Analysis Attacks on Smartcards. In *WOST '99: Proceedings of the USENIX Workshop on Smartcard Technology*, pages 151–162, Berkeley, CA, USA, 1999. USENIX Association.

[23] National Bureau of Standards. Data Encryption Standard. Federal Information Processing Standard #46, 1977.

[24] National Institute of Standards and Technology. Advanced Encryption Standard (AES). Federal Information Processing Standard #197, 2001.

[25] Roman Novak. Side-Channel Attack on Substitution Blocks. In Jianying Zhou, Moti Yung, and Yongfei Han, editors, *Applied Cryptography and Network Security – ACNS '03*, volume 2846 of *Lecture Notes in Computer Science*, pages 307–318. Springer-Verlag, 2003.

[26] Roman Novak. Sign-Based Differential Power Analysis. In Kijoon Chae and Moti Yung, editors, *Workshop on Information Security Applications – WISA '03*,

volume 2908 of *Lecture Notes in Computer Science*, pages 203–216. Springer, 2003.

[27] Pascal Paillier and Ingrid Verbauwhede, editors. *Cryptographic Hardware and Embedded Systems – CHES '07, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, 2007. Springer-Verlag.

[28] Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, 2011. Springer.

[29] Denis Réal, Vivien Dubois, Anne-Marie Guilloux, Frédéric Valette, and M'hamed Drissi. SCARE of an Unknown Hardware Feistel Implementation. In Gilles Grimaud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Application – CARDIS '08*, volume 5189 of *Lecture Notes in Computer Science*, pages 218–227. Springer, 2008.

[30] Matthieu Rivain and Thomas Roche. SCARE of Secret Ciphers with SPN structures. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT '13*, volume 8269 of *Lecture Notes in Computer Science*, pages 526–544. Springer-Verlag, 2013.

[31] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A New Class of Collision Attacks and Its Application to DES. In Thomas Johansson, editor, *Fast Software Encryption – FSE '03*, volume 2887 of *Lecture Notes in Computer Science*, pages 206–222. Springer-Verlag, 2003.

[32] Kai Schramm, Gregor Leander, Patrick Felke, and Christof Paar. A Collision-Attack on AES: Combining Side Channel- and Differential-Attack. In Joye and Quisquater [15], pages 163–175.

[33] Sung-Ming Yen and Marc Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Transactions on Computers*, 49(9):967–970, 2000.