FIRE: Fault Injection for Reverse Engineering

Manuel San Pedro[1], Mate Soos[2], and Sylvain Guilley[1]

[1] Institut TELECOM
[2] INRIA, Security Research Labs

Abstract. In this paper, we propose a new technique that uses fault injection to reverse-engineer a private block cipher implemented with an unknown S-box. The private algorithm we wish to retrieve differs from a known algorithm in the choice of the S-Box, which we find using a novel, fault-injecting technique. The main idea is to consider the components of the S-Box as the solutions of a linear boolean system, whose equations stem from the faults injected, using existing fault models. We focus on two well-known block ciphers, DES and AES, and prove it to be feasible to retrieve the the S-Box for both cases. We present the fault models used, the equations extracted from the faults injected, and analyse the final results. Given the detailed analysis, the technique can be applied with ease to most ciphers employing an S-box.

1 Introduction

According to Kerckhoffs's principle, a cryptosystem should be secure even if everything about the system except the secret key is public knowledge [9]. Even though this became a fundamental principle of modern cryptology, it is moderately common for companies and sometimes even standards bodies to keep the inner workings of a system secret [1,6]. We then talk about *security through obscurity*, or *black-box cryptography*.

Under Kerckhoffs's principle, cryptanalysis consists in retrieving the cipher key. But when dealing with security through obscurity, the goal is now modified to also retrieve information on the private algorithm. This is called reverse-engineering. Nowadays, with the omnipresence of embedded cryptography, it has become crucial to be able to perform attacks on electronic devices embedding unknown cryptosystems.

Previous attempts at reverse-engineering unknown cryptosystems were either through (electro-)optical means, such as the discovery of the MIFARE algorithm [6], or through the use of side-channel analysis [11]. Side-channel analysis was originally devised to find the secret key through the measurement of physical characteristics of the chip such as power intake. Guilley et al. [7] employed this technique to retrieve the internals of black-box ciphers. This is called the side-channel analysis for reverse-engineering (SCARE) attack.

In this paper, we present a new type of attack employing the principle of fault injection [2] to retrieve the unknown S-box of a black-box cipher. Fault injection was originally devised to retrieve the secret key through injection of faults into

C.A. Ardagna and J. Zhou (Eds.): WISTP 2011, LNCS 6633, pp. 280–293, 2011.

the chip executing the algorithm and observing the modified output. Our attack injects faults into the chip, collects the output from the chip, performs analysis of this data and finally converts the data into a set of equations in binary variables, which are finally solved using Gaussian elimination to retrieve the S-box. This new type of attack we call fault injection for reverse engineering (FIRE).

The rest of the paper is organised as follows. In Sect. 2, we describe the state of the art, such as physical attacks of cryptosystems and linear systems solving. In Sect. 3, we present a DES-based cryptosystem, and a FIRE attack on it. Then, in Sect. 4, we describe an AES-based cryptosystem, and its corresponding FIRE attack. Finally, in Sect. 5 we conclude this paper.

2 State of the Art

2.1 Physical Attacks on Cryptographic Systems

Most of the cryptographic algorithms used in serious applications are supposed to be secure against algorithmic attacks. However, they are implemented on physical components, and hence become vulnerable against physicals attacks. Once such algorithms are implemented, either on dedicated hardware or as software on a micro-controller, the different physical properties of the algorithm can be observed. Over the years, sophisticated attacks have been developed to attack cryptographic devices through such observations.

Side-channel attack. The physical implementation of a cipher may reveal useful information about the secret key in an indirect way. Kocher in [10] and in [11] published two novel attack techniques exploiting side channel leakage of cryptographic devices. Computation requires time, consumes power and causes electromagnetic radiations: all these are possible sources of information related to the secret key. These techniques are powerful, as they allow to reduce the complexity of a brute-force attack by several orders of magnitude. However, they require physical access to the device to collect the necessary measurements.

Fault-injection attack. Fault attacks is the active way of attacking the physical implementation of an algorithm. During the proper functioning of the device, the attacker perturbs it by injecting hardware faults which produce an erroneous (or *faulted*) output. The attacker then exploits this to retrieve secret information. As explained in [8], the most common ways to carry out such an attack are manipulating the supply voltage or the the external clock, or applying laser or X-ray beams.

The SCARE attack. More recently it has been shown ([3,5,7]) that side-channel attacks could be used to retrieve secret parts of private algorithms. This is called side-channel attack for reverse-engineering, or simply SCARE. when a side-channel is used to retrieve an S-box on a private block cipher such as DES or AES, the attacker studies the transition $y = \mathcal{SB}(x \oplus k)$. In a classical side-channel attack, $\mathcal{SB}$ is known and we wish to retrieve k. In SCARE, we assume to know k and wish to retrieve $\mathcal{SB}$.

2.2 Solving Linear Boolean Systems

If we consider an S-box as a boolean function $f_{n\to m}$ (i.e. a boolean function from $\{0,1\}^n$ to $\{0,1\}^m$), we can split it into m and $f_{n\to 1}$, called the *components*. Each one of the components will be considered as a vector $s \in \{0,1\}^{2^n}$, being the solution of a linear system in 2^n variables. Each one of the faults injected brings a certain l number of equations (depending on the fault model), that the component s must satisfy. This means that each component s is one of the solutions of the system in $\{0,1\}$ of l equations:

$$A \cdot X = B. \tag{1}$$

where A is a $l \times 2^n$ boolean matrix, and both X and B are vectors of 2^n elements.

The equations are of the form $\bigoplus_{i=0}^{2^n-1} a_i \cdot x_i = b_i$. Let $\mathcal{L}$ be the set of solutions of the system $\mathcal{L} = \{s \in \{0,1\}^{2^n} : A \cdot s = B\}$. Let us note that

$$s \in \mathcal{L} \Leftrightarrow \bar{s} \in \mathcal{L} \tag{2}$$

It stems from the fact that if α and β are boolean variables, then $\alpha \oplus \beta = \bar{\alpha} \oplus \bar{\beta}$. This property will be important for the rest of the study, since the minimum of candidates returned will be 2. To solve this linear system of equations, we have used the Sage software [14] to perform the Gaussian elimination, but any mathematical software is adequate for the job, as the matrices are typically quite small.

3 The Case of DES

We first give a description of a FIRE attack on a DES-like cryptosystem. Even if the attack has already been shown by Biham & Shamir in [2], it gives us a good foundation to proceed during the more complex case of a SPN such as AES in Sect. 4.

The Data Encryption Standard (DES) was developed in the 1970s by the National Bureau of Standards with the help of the National Security Agency. Its purpose was to provide a standard method for protecting sensitive commercial and unclassified data. IBM created the first draft of the algorithm, calling it LUCIFER. DES officially became a federal standard in November of 1976 [12]. DES is a symmetric cryptosystem, specifically a 16-round Feistel cipher. It has a 64-bit block size and uses a 56-bit key. From this key, 16 sub-keys are created and are used at each round. The input is split in two halves. The progression of the cipher is described in Fig. 1.

The round function, applied to a 32 bits register R and a 48 bits round Key K, $F(R,K)$, consists in the succession of 4 sub-functions: first, E is an expansion function applied to R which returns a 48 bits output. The key K is then XOR-ed to $E(R)$. S is the substitution function. It consists in 8 S-Boxes $\mathcal{SB}_0, \ldots, \mathcal{SB}_7$ each of which map a 6-bit input to a 4-bit output. A 32-bit permutation P is finally applied to the output of S.

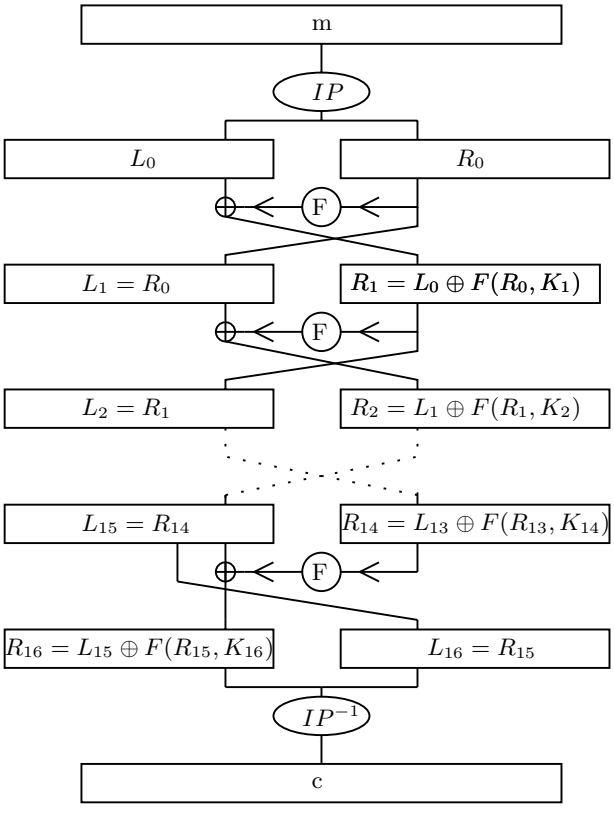

Fig. 1. The DES cipher, a 16-round Feistel cipher. IP is a 64 bit permutation. The round function applies F to the right half of the register, XORs the result to the left half, and exchanges the roles of the halves.

We consider the fault model introduced by Biham and Shamir in [2]: it assumes that the attacker is able to inject faults at the last round, round no. 15, on the right register R_{15}. We consider that the substitution function S has been modified and kept secret. We then wish to retrieve $\mathcal{SB}_0, \ldots, SB_7$, the 8 S-Boxes which compose it.

Let $c = (L_{16}, R_{16})$ be the correct and $c^\star = (L^\star_{16}, R^\star_{16})$ be the faulty ciphertext, resulting from the same plaintext m and secret key K. If we consider that the secret key is not known but fixed to a certain value, we will not retrieve the exact S-boxes, instead we will retrieve the function $x \mapsto \mathcal{SB}_i(x \oplus k_i)$, where k_i is key input of the i^{th} S-Box.

Without loss of generality, let us consider that the key is known for the attack, hence we can ignore it for our present discussion. We thus have:

$$\begin{aligned} & R_{16} = L_{15} \oplus F(R_{15}) = L_{15} \oplus F(L_{16}) \\ \text{and} \quad & R^\star_{16} = L_{15} \oplus F(R^\star_{15}) = L_{15} \oplus F(L^\star_{16}). \end{aligned}$$

hence we get:

$$R_{16} \oplus R_{16}^{\star} = F(L_{16}) \oplus F(L_{16}^{\star})$$
$$R_{16} \oplus R_{16}^{\star} = P[S(E(L_{16}))] \oplus P[S(E(L_{16}^{\star}))]$$
$$P^{-1}[R_{16} \oplus R_{16}^{\star}] = S(E(L_{16})) \oplus S(E(L_{16}^{\star})).$$

Since c and $c^{\star}$ are known, the only unknown register, L_{15}, disappears once R_{16} is XOR-ed with $R_{16}^{\star}$. The intrinsic design of Feistel block-ciphers allows us to have the knowledge of the fault injected, and its effect during the cipher, giving us the difference at the input and output of the S-Boxes. We note Δ_{in} and Δ_{out}, those differences:

$$\Delta_{in} = E(L_{16}) \oplus E(L_{16}^{\star})$$
$$\Delta_{out} = P^{-1}[R_{16} \oplus R_{16}^{\star}].$$

where Δ_{in} and Δ_{out} are 48 and 32 bits long. However, if we focus on the i^{th} S-box Sb_i for instance, we can consider Δ_{in}^{i} and Δ_{out}^{i} as 6 and 4 bits long. We know $x = E(L_{16})[6 * i : 6 * (i + 1)]$, the 6 bits input of $\mathcal{SB}_i$ during the unaltered cipher, $x^{\star} = E(L_{16}^{\star})[6 * i : 6 * (i + 1)]$, the 6 bits input of $\mathcal{SB}_i$ during the faulty cipher. We have the relation:

$$Sb_i(x) \oplus Sb_i(x^{\star}) = \Delta_{out}^{i}.$$

Our goal is to retrieve Sb_i, which is a boolean function from $\{0,1\}^6$ to $\{0,1\}^4$. Let's consider it component-wise, i.e. as 4 functions from $\{0,1\}^6$ to

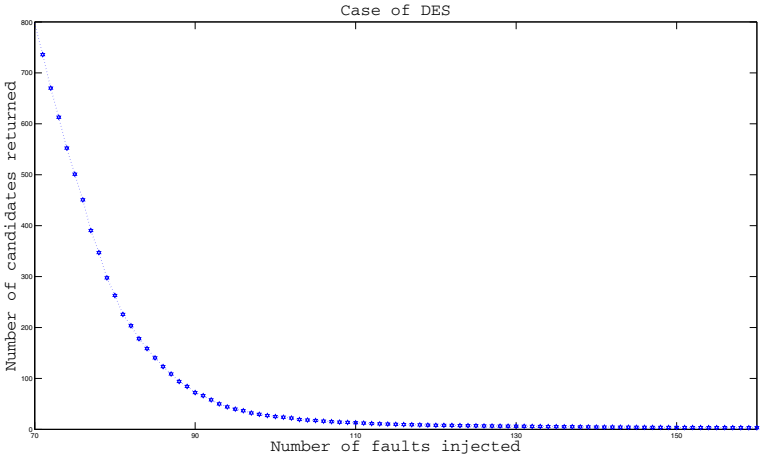

Fig. 2. Attacking component 0 of the first S-Box of DES: On the x axis, the number of faults injected, on the y-axis the mean of $\#\mathcal{L}_{1,0,x}$ after 1000 tries. In the end, we only have 2 candidates.

$\{0,1\}$: (s_0, s_1, s_2, s_3). From each injected fault, we must have:

$$\text{For } j = 0, \ldots, 3, \quad s_{j,x} \oplus s_{j,x^\star} = \Delta^i_{out}(j), \tag{3}$$

where $\Delta^i_{out}(j)$ is the j^{th} bit of Δ^i_{out}. For each injected fault and for each component j, s_j must satisfy the previous equation. It is then added to the final system. We now have a distinguisher, we can define $\mathcal{L}_{i,j,N}$ as the set of candidates for the j^{th} component of the i^{th} S-box. Considering N fault injections, giving us $(x_k, x^\star_k, \Delta_{out,k})$ (k from 1 to N), we have:

$$\mathcal{L}_{i,j,N} = \{s \in \{0,1\}^{64} \text{ such that } \forall k, k \leq N : s_{x_k} \oplus s_{x^\star_k} = \Delta^i_{out,k}(j)\}.$$

Simulating an error perturbing randomly one single input bit of an S-box of DES, we reach the final set of two candidates mentioned at eq. (4) after approximately 130 fault injections. Fig. 2 illustrates the mean progression of $\#\mathcal{L}_{1,0,N}$ with 1000 experiments. This attack converges to the expected solution, meaning that, since we have the property (2),

$$\exists n_0 \text{ such that } \forall n > n_0, \quad \mathcal{L}_{i,j,n} = \{s_j, \bar{s_j}\}. \tag{4}$$

Note that in order to fully retrieve the 8 S-boxes, one has to test both candidates for all the 32 components. This leads to an exhaustive search in 2^{32}, which is trivially feasible.

4 The Case of AES

AES is a widely used symmetric-key encryption by Daemen and Rijman [4], adopted as a standard by the National Institute of Standards and Technology of the US. It is based on a design principle known as a Substitution Permutation Network (SPN). AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits. It operates on a 4×4 array of bytes, termed the state (where 1 byte = 8 bits). Most calculations carried out by the cipher are done in the finite field of $\mathcal{GF}(2^8)$.

The AES cipher is specified as a number of repetitions of transformation rounds, each round made up with 4 round transformations: SubBytes, MixColumns, ShiftRows and AddRoundKey. Note that the last round is exempt from MixColumns.

Without loss of generality, since we consider that the cipher key is known, we set it to 0, and we also discard the final ShiftRows operation since it can trivially be inverted. Hence we only consider operations MixColumns and SubBytes, as explained below in detail.

MixColumns applies a linear transformation to a column of the state:

$$\text{MixColumns}\left(\begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix}\right) = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix},$$

where the operations are performed in $\mathcal{GF}(2^8)$.

SubBytes is a non linear transformation which is applied to each byte of the state. It is traditionally implemented as a S-box, which can be seen as a boolean function $\mathcal{SB}$ from 8 bits to 8 bits. Note that SubBytes is a bijection.

$$\text{SubBytes}\left(\begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix}\right) = \begin{bmatrix} \mathcal{SB}(x) \\ \mathcal{SB}(y) \\ \mathcal{SB}(z) \\ \mathcal{SB}(t) \end{bmatrix}.$$

In our attack, this function is unknown, and the goal is to retrieve it.

4.1 Fault Injection

Let us assume that we are able to inject a fault on one byte of the block, just before the last MixColumns, during the 9^{th} round. The attack is column-wise, meaning that we only care about the column on which the fault is injected. For example, let us look at the first column of a regular cipher, from the last MixColumns until the end of the cipher. We have:

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \xrightarrow{\text{MC}} MC\left(\begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix}\right) \xrightarrow{\text{SB}} \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} = c. \tag{5}$$

Now, the same data is processed, but with a fault ϵ injected before the last MixColumns. Fig. 3 illustrates the propagation of the error.

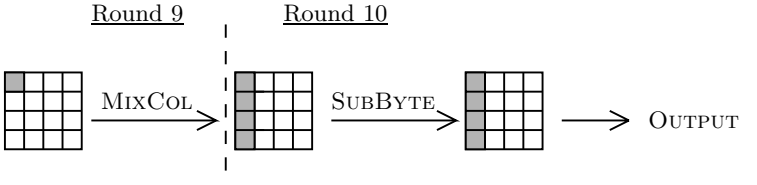

Fig. 3. Propagation of the fault on our simplified AES: we perturb a byte just before the last MixColumns. The error propagates to the whole column.

We thus have:

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \xrightarrow{\text{FI}} \begin{bmatrix} \alpha \oplus \epsilon \\ \beta \\ \gamma \\ \delta \end{bmatrix} \xrightarrow{\text{MC}} MC\left(\begin{bmatrix} \alpha \oplus \epsilon \\ \beta \\ \gamma \\ \delta \end{bmatrix}\right) \xrightarrow{\text{SB}} \begin{bmatrix} x^\star \\ y^\star \\ z^\star \\ t^\star \end{bmatrix} = c^\star. \tag{6}$$

Now that we have a triplet $(c, c^\star, \epsilon)$. Let us examine how we could exploit Fault Injection to extract information on $\mathcal{SB}$. We start the attack from the ciphertexts, we retrieving $\mathcal{SB}^{-1}$, which is exactly the same since $\mathcal{SB}$ is bijective in SPNs.

We have, from eq. (5) and eq. (6):

$$\mathcal{SB}^{-1}(c) \oplus \mathcal{SB}^{-1}(c^\star) = MC\left(\begin{bmatrix}\alpha\\ \beta\\ \gamma\\ \delta\end{bmatrix}\right) \oplus MC\left(\begin{bmatrix}\alpha \oplus \epsilon\\ \beta\\ \gamma\\ \delta\end{bmatrix}\right)$$
$$= MC\left(\begin{bmatrix}\epsilon\\ 0\\ 0\\ 0\end{bmatrix}\right) = \begin{bmatrix}02 \cdot \epsilon\\ \epsilon\\ \epsilon\\ 03 \cdot \epsilon\end{bmatrix},$$

because MixColumns is linear. It translates into the system

$$\begin{array}{ll}\mathcal{SB}^{-1}(x) \oplus \mathcal{SB}^{-1}(x^\star) & = 02 \cdot \epsilon\\ \mathcal{SB}^{-1}(y) \oplus \mathcal{SB}^{-1}(y^\star) & = \epsilon\\ \mathcal{SB}^{-1}(z) \oplus \mathcal{SB}^{-1}(z^\star) & = \epsilon\\ \mathcal{SB}^{-1}(t) \oplus \mathcal{SB}^{-1}(t^\star) & = 03 \cdot \epsilon\end{array}. \tag{7}$$

4.2 Translation of the FI into Equations

Let us remind ourselves that $\mathcal{SB}^{-1}$ is a boolean function from $\{0,1\}^8$ to $\{0,1\}^8$. Considering it component-wise, i.e. as 8 independent functions from $\{0,1\}^8$ to $\{0,1\}$:

$$\mathcal{SB}^{-1} = \{\mathcal{SB}_0^{-1}, \mathcal{SB}_1^{-1}, \ldots, \mathcal{SB}_7^{-1}\} \quad \text{with} \quad \mathcal{SB}_i^{-1} : \{0,1\}^8 \mapsto \{0,1\}\}.$$

Now, $\mathcal{SB}_i^{-1}$ can be seen as a set of 256 boolean variables:

$$\mathcal{SB}_i^{-1} = \{s_{i,0}, s_{i,1}, \ldots, s_{i,255}\}.$$

If we consider bit-wise the equations given in (7) then for a fault injected, we know that, necessarily, for $i = 0 \ldots 7$, $\mathcal{SB}_i^{-1}$ has to satisfy

$$\begin{array}{ll}s_{i,x} \oplus s_{i,x^\star} & = (02 \cdot \epsilon)_i\\ s_{i,y} \oplus s_{i,y^\star} & = \epsilon_i\\ s_{i,z} \oplus s_{i,z^\star} & = \epsilon_i\\ s_{i,t} \oplus s_{i,t^\star} & = (03 \cdot \epsilon)_i\end{array}. \tag{8}$$

These four equations are to be manipulated according to the fault model, and used to build the final system that is solved with Gaussian elimination to finally give the solutions.

4.3 Random and unknown Faults

First, we discuss the fault model that is close to the one presented by Piret and Quisquater in [13]. The error is injected on the first byte of the state, just before

the last MixColumns. It is random and unknown. By adding lines of the system (7), without any knowledge of the value of ϵ, we have:

$$\begin{array}{l}\mathcal{SB}^{-1}(x) \oplus \mathcal{SB}^{-1}(x^\star) \oplus \mathcal{SB}^{-1}(y) \oplus \mathcal{SB}^{-1}(y^\star) \oplus \mathcal{SB}^{-1}(t) \oplus \mathcal{SB}^{-1}(t^\star) = 0 \\ \mathcal{SB}^{-1}(x) \oplus \mathcal{SB}^{-1}(x^\star) \oplus \mathcal{SB}^{-1}(z) \oplus \mathcal{SB}^{-1}(z^\star) \oplus \mathcal{SB}^{-1}(t) \oplus \mathcal{SB}^{-1}(t^\star) = 0\end{array},$$

since $03 \cdot \epsilon \oplus 02 \cdot \epsilon \oplus \epsilon = 0$. The operations are made on $\mathcal{GF}(2^8)$.

Each one of the 8 components of $\mathcal{SB}^{-1}$ has to satisfy these equations. Now that we have removed ϵ, we can inject them into the system. Once solved, this system returns all the satisfying candidates, including the eight solutions. Considering N fault injections, giving us $(c_k, c_k^\star)$ (k from 1 to N), we can define the distinguisher $\mathcal{L}_N$ for the attack of $\mathcal{SB}^{-1}$:

$$\mathcal{L}_N = \left\{ s \in \{0,1\}^{256} \text{ such that } \forall k < N, \begin{array}{l} s_{x_k} \oplus s_{x_k^\star} \oplus s_{y_k} \oplus s_{y_k^\star} \oplus s_{t_k} \oplus s_{t_k^\star} = 0 \\ s_{x_k} \oplus s_{x_k^\star} \oplus s_{z_k} \oplus s_{z_k^\star} \oplus s_{t_k} \oplus s_{t_k^\star} = 0 \end{array}, \right\}.$$

It so happens that after $n_0 \approx 400$ faults injected, we have a constant set of solutions $\mathcal{S}$:

$$\forall n > n_0, \quad \mathcal{L}_n = \mathcal{S}.$$

More precisely, the attack converges to a set $\mathcal{S}$ with 512 candidates. First we describe in detail this set $\mathcal{S}$, and then we discuss the possible conclusion of the attack through exhaustive search.

To account for the 512 solutions, we consider $\mathcal{S}$ as an *orbit* of the 8 components of $\mathcal{SB}^{-1}$: we have always $\mathcal{SB}_0^{-1}, \mathcal{SB}_1^{-1}, \ldots, \mathcal{SB}_7^{-1} \in \mathcal{S}$. But we also have $(0,0,\ldots,0)$ and $(1,1,\ldots,1)$ in $\mathcal{S}$ (they indeed satisfy all the equations brought by the distinguisher), we then state that:

Proposition 1. $u, v \in \mathcal{S} \Rightarrow u \oplus v \in \mathcal{S}$.

Proof. Without loss of generality, we shorten the definition of $\mathcal{S}$ to a single boolean equation, which does not change with the real context. For instance:

$$\mathcal{S} = \{s \in \{0,1\}^{256} \text{ such that } \quad s_y \oplus s_y^\star \oplus s_z \oplus s_z^\star = 0\}.$$

Now let $u, v \in \mathcal{S}$.

$$\begin{array}{r} u_y \oplus u_{y^\star} \oplus u_z \oplus u_{z^\star} = 0 \text{ , and } v_y \oplus v_{y^\star} \oplus v_z \oplus v_{z^\star} = 0. \\ \text{Then } u_y \oplus u_{y^\star} \oplus u_z \oplus u_{z^\star} \oplus v_y \oplus v_{y^\star} \oplus v_z \oplus v_{z^\star} = 0. \\ \text{Then } (u \oplus v)_y \oplus (u \oplus v)_{y^\star} \oplus (u \oplus v)_z \oplus (u \oplus v)_{z^\star} = 0. \\ \text{Finally } u \oplus v \in \mathcal{S}. \end{array}$$

We now can define $\mathcal{S}$ such that:

$$\mathcal{S} = \{a_0 \cdot \mathcal{SB}_0^{-1} \oplus \ldots a_7 \cdot \mathcal{SB}_7^{-1} \oplus a_8 \cdot (1,\ldots,1), \quad a_i \in \{0,1\}\}.$$

We can remove from $\mathcal{S}$ the trivial solution $(1,\ldots,1)$ and $(0,\ldots,0)$: in fact, it is mandatory for a SPN S-box to be bijective, and it would not be the case if $(1,\ldots,1)$ or $(0,\ldots,0)$ was one of the components.

From this set, how can the full S-box be efficiently retrieved? We have 510 candidates that must be replaced into the correct position out of 8 possible choices. A naive exhaustive search would lead to $\mathcal{C}_8^{510} \times 8! \approx 2^{71}$ possibilities.

However, as we have already noticed, $\forall s \in \mathcal{S}, \bar{s} \in \mathcal{S}$. We can form 255 groups of elements of $\mathcal{S}$, each of them including a candidate and its complement. For an optimal exhaustive search, one has to select 8 of those groups, and then test the 256 possibilities. This would lead to $2^8 \times \mathcal{C}_8^{255} \approx 2^{57}$ possibilities to finish the attack. This computational complexity is moderately high, but can be achieved with a large set of modern GPUs and/or FPGAs, and is not out of reach of any major organisation such as multinational companies or governments. However, we also propose another solution by finishing the attack using the SCARE method.

4.4 SCARE Conclusion of a FIRE Attack

In this section, we propose a finishing of a FIRE attack when we are in the context described in Sect. 4.3. We have a set $\mathcal{S}$, of 510 candidates containing the 8 component of $\mathcal{SB}^{-1}$.

In order to use side-channel information to finish the attack, we use the curves of the DPA-Contest [15] to find $\mathcal{SB}^{-1}$. The context is the following. We have

- N power traces corresponding of the functioning of the components with known inputs/outputs/cipher keys.
- The set $\mathcal{S}$ of a reduced amount of candidates for the components of $\mathcal{SB}^{-1}$. Here, 510.

It is well-known that the power consumption of components strongly depend on the data processed, and more exactly the number of bit-flips completed. This number is given by the hamming distance between a register at a time t and $t+1$. We then talk about Hamming distance model. We study here the transition during the last SubBytes of the AES chiper.

For every candidates $s \in \mathcal{S}$, for every component j of $\mathcal{SB}^{-1}$, we compute what would be the hamming distance between c (which is known) and the state at the input of the last SubBytes, if we would have $s = \mathcal{SB}_j^{-1}$. We then use a distinguisher (Pearson's correlation) in order to measure the dependence between those hamming distances and the power traces.

On Fig. 4, the correlation traces resulting from the attack of the 7^{th} component by using SCARE. This means that we are looking for $\mathcal{SB}_7^{-1}$ amongst the 510 members of $\mathcal{S}$. On the figure, by using 10000 traces, we clearly can identify $\mathcal{SB}_7^{-1}$ in red and bold, $\overline{\mathcal{SB}_7^{-1}}$, the symmetric below, and the 508 bad candidates, giving a correlation close to zero. An adversary able to perform fault injections on a component is very likely to be able to get a campaign of acquisition of power traces in order to conclude the attack this way. Hence it alleviates the burden on the attacker of making the exhaustive search in 2^{57} as munitioned at the end of Sec. 4.3.

Note that the SCARE attack is feasible here since we have a very restricted number of candidate for the solutions. When dealing with SCA, the number

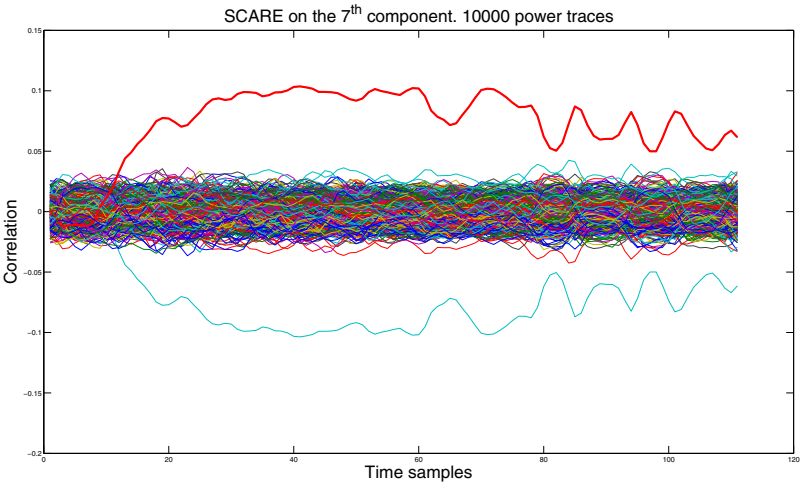

Fig. 4. SCARE on the 7^{th} component of $\mathcal{SB}^{-1}$, with $N = 10000$ power traces. We have the 510 correlation traces: on the x-axis the time samples of the power traces, on the y-axis, the value of the correlation. We clearly identify here the solution (on the top) and its complementary (on the bottom)

of candidates to test is very important: 256 hypothesis to test when we want to retrieve a key byte, but 2^{2^8} hypothesis to test when we are looking for a single component of $\mathcal{SB}^{-1}$. Here the FIRE attack carried out most of the job by reducing the 2^{2^8} to 510.

On Fig. 4, we have the results with $N = 10000$ power curves. However, from $N = 5000$ curves (taken randomly from the ones available for the DPA Contest), the attack is feasible, meaning that we are able to extract the solutions.

4.5 Results with Various Fault Models and Contexts

In this section, we present several other realistic fault models, or context allowing us to perform a FIRE attacks.

Random and known faults. Let us consider the strongest fault model: we are able to inject a random and known fault during the cipher execution.

The advantage with this model, is that, since we know ϵ, we are able to target which one of the components of $\mathcal{SB}^{-1}$ we are attacking.

Considering N fault injections, giving us $(c_k, c_k^\star, \epsilon^k)$, $k = 1, \dots N$, we can define the distinguisher $\mathcal{L}_{i,N}$ for the attack of the i^{th} component of $\mathcal{SB}^{-1}$:

$$\mathcal{L}_{i,N} = \left\{ s \in \{0,1\}^{256} \text{ such that } \forall k < N, \quad \begin{array}{ll} s_{x_k} \oplus s_{x_k^\star} & = (02 \cdot \epsilon^k)_i \\ s_{y_k} \oplus s_{y_k^\star} & = \epsilon_i^k \\ s_{z_k} \oplus s_{z_k^\star} & = \epsilon_i^k \\ s_{t_k} \oplus s_{t_k^\star} & = (03 \cdot \epsilon^k)_i \end{array} \right\}.$$

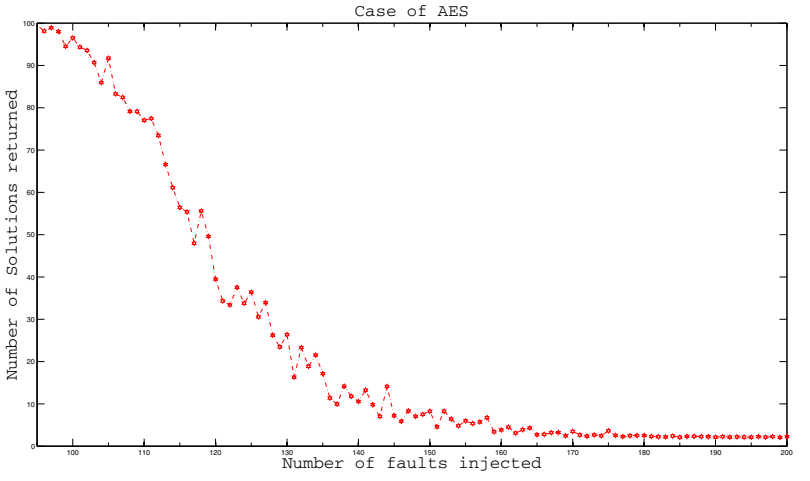

Fig. 5. Attacking component 0 of AES S-Box inverse. On the x-axis, the number of faults injected, on the y-axis the mean of $\#\mathcal{L}_{0,x}$ after 100 tries. In the end, we only have 2 candidates.

This model, combined with the technique described in Sect. 4.2, allows us to retrieve the full $\mathcal{SB}^{-1}$ in less than 180 faults injected. Fig. 5 illustrates the progression of $\mathcal{L}_{0,N}$, simulating an error occurring randomly on the first byte of the state just before the last MixColumns.

Stuck-at model. It has been shown that it is possible for an attacker to force a byte to a certain value, that it can choose. If we suppose that, just before the last SubBytes, one can force the first byte to a given value τ:

$$\begin{bmatrix}\alpha\\ \beta\\ \gamma\\ \delta\end{bmatrix} \xrightarrow{\text{FI}} \begin{bmatrix}\tau\\ \beta\\ \gamma\\ \delta\end{bmatrix} \xrightarrow{\text{SB}} \begin{bmatrix}x^\star\\ y^\star\\ z^\star\\ t^\star\end{bmatrix} = c^\star.$$

Hence he has access to $x^\star$ which is equal to $\mathcal{SB}(\tau)$. It leads to a trivial attack, since with 256 accurate *stuck-at* injections, one can retrieve the full S-Box.

Note that even one single *stuck-at* injection, we get a lot of information to bring into the system, if we decide do use different models during the attack.

In the case where it is not possible to inject a *stuck-at* fault at the input of the last SubBytes, but that it can be done just before the last MixColumns:

$$\begin{bmatrix}\alpha\\ \beta\\ \gamma\\ \delta\end{bmatrix} \xrightarrow{\text{FI}} \begin{bmatrix}\tau\\ \beta\\ \gamma\\ \delta\end{bmatrix} \xrightarrow{\text{MC}} MC\left(\begin{bmatrix}\tau\\ \beta\\ \gamma\\ \delta\end{bmatrix}\right) \xrightarrow{\text{SB}} \begin{bmatrix}x^\star\\ y^\star\\ z^\star\\ t^\star\end{bmatrix} = c^\star.$$

In that case, even if we know the value of τ, α is assumed to be random. But we have:

$$\exists \epsilon \in \mathcal{GF}(2^8) \text{ such that } \tau = \alpha \oplus \epsilon.$$

It hence leads to the fault model presented at Sec. 4.3, just as if we would have injected an unknown and random ϵ.

5 Conclusion

In this paper, we have introduced a new tool to reverse-engineer a private algorithm. This new FIRE attack allows us to retrieve the S-Box of private block-ciphers in a reasonable number of faults injected and under plausible and existing fault models. For the sake of practical demonstration, we have carried out the attack on two major ciphers, AES and DES, but the attack can be made to work on almost any cipher containing and unknown S-Box. In the case of the DES S-boxes, around 1000 Fault Injections are needed and a final exhaustive search in 2^{32} is necessary to fully retrieve all the 8 S-Boxes. For AES, under the most plausible model, around 400 fault injections suffice and lead to a finite set of 510 candidates. We can then either conclude the attack using exhaustive search in 2^{57}, or perform a data acquisition campaign and finish the attack using SCARE.

References

1. Anderson, R.: A5 (was: Hacking digital phones). Newsgroup Communication (1994)
2. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
3. Clavier, C.: An improved SCARE cryptanalysis against a secret A3/A8 GSM algorithm. In: McDaniel, P., Gupta, S.K. (eds.) ICISS 2007. LNCS, vol. 4812, pp. 143–155. Springer, Heidelberg (2007)
4. Daemen, J., Rijmen, V.: The block cipher rijndael. In: Schneier, B., Quisquater, J.-J. (eds.) CARDIS 1998. LNCS, vol. 1820, pp. 277–284. Springer, Heidelberg (2000)
5. Daudigny, R., Ledig, H., Muller, F., Valette, F.: Scare of the des. In: ACNS, pp. 393–406 (2005)
6. Garcia, F.D., de Koning Gans, G., Muijrers, R., van Rossum, P., Verdult, R., Schreur, R.W., Jacobs, B.: Dismantling MIFARE classic. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 97–114. Springer, Heidelberg (2008)
7. Guilley, S., Sauvage, L., Micolod, J., Réal, D., Valette, F.: Defeating Any Secret Cryptography with SCARE Attacks. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 273–293. Springer, Heidelberg (2010)
8. Hamid, H. B.-E., Choukri, H., Tunstall, D. N. M., and Whelan, C. The sorcerer's apprentice guide to fault attacks
9. Kerckhoffs, A.: La cryptographie militaire. Journal des Sciences Militaires IX, 5–83 (1883)
10. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems, pp. 104–113

11. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
12. National Bureau of Standards. Data Encryption Standard (1977)
13. Piret, G., Quisquater, J.-J.: A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 77–88. Springer, Heidelberg (2003)
14. Stein, W., et al.: Sage Mathematics Software, `http://www.sagemath.org`
15. VLSI Research Group TELECOM ParisTech. The DPA contest (2008/2009), `http://www.dpacontest.org/`