DFA on LS-Designs with a Practical Implementation on SCREAM

Benjamin Lac, Anne Canteaut, Jacques Jean-Alain Fournier, Renaud Sirdey

▶ **To cite this version:**

Benjamin Lac, Anne Canteaut, Jacques Jean-Alain Fournier, Renaud Sirdey. DFA on LS-Designs with a Practical Implementation on SCREAM. COSADE 2017 - Constructive Side-Channel Analysis and Secure Design, Apr 2017, Paris, France. pp.223–247, 10.1007/978-3-319-64647-3_14 . hal-01649974

HAL Id: hal-01649974

https://inria.hal.science/hal-01649974

Submitted on 28 Nov 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DFA on LS-Designs with a Practical Implementation on SCREAM

Benjamin Lac[1,5], Anne Canteaut[2], Jacques Fournier[3], and Renaud Sirdey[4]

[1] CEA-Tech, Gardanne, France,
[2] Inria, Paris, France,
[3] CEA-Leti, Grenoble, France,
[4] CEA-List, Saclay, France,
[5] ENSM-SE, Saint-Étienne, France,
{benjamin.lac, jacques.fournier, renaud.sirdey}@cea.fr,
anne.canteaut@inria.fr

Abstract. LS-Designs are a family of SPN-based block ciphers whose linear layer is based on the so-called interleaved construction. They will be dedicated to low-end devices with high performance and low-resource constraints, objects which need to be resistant to physical attacks. In this paper we describe a complete Differential Fault Analysis against LS-Designs and also on other families of SPN-based block ciphers. First we explain how fault attacks can be used against their implementations depending on fault models. Then, we validate the DFA in a practical example on a hardware implementation of SCREAM running on an FPGA. The faults have been injected using electromagnetic pulses during the execution of SCREAM and the faulty ciphertexts have been used to recover the key's bits. Finally, we discuss some countermeasures that could be used to thwart such attacks.

Keywords: Lightweight cryptography · DFA · SPN-based block ciphers · LS-Designs · SCREAM · EM fault attacks.

1 Introduction

The advent of the Internet of Things (IoT) has brought the need for new cryptographic primitives to suit the high performance, low power and low resource constraints of IoT devices. Ciphers like AES, which are good enough for embedded devices like smart cards, do not satisfy the constraints of devices like RFID tags or nodes in sensor networks. During the past years, several lightweight block ciphers have been proposed, some are highly efficient software-oriented ciphers like PRIDE [3] or SPECK [5], and some are rather highly efficient hardware-oriented ciphers like PRESENT [10], PRINCE [12] or SIMON [5]. In terms of security, these ciphers are mainly designed to resist black-box mathematical attacks. However, since they are used in IoT devices in pervasive environments, we ought to also look at implementation-related attacks. Indeed, resistance against side channel attacks is now considered as a valuable property which should be taken in consideration when designing lightweight ciphers as underlined by the ciphers FIDES [8], PICARO [31] and Zorro [16]. In that respect, several physical attacks have been proposed against lightweight ciphers. One of them is a complete Differential Fault Analysis (DFA) which exploits the design of the linear layer introduced on PRINCE [37] and on PRIDE [1,25]. DFA is a particular

physical attack, in which we compare the results of a correct computation to one which has been disturbed at a precise time, in order to infer information about the key bits used in the cipher. In this paper, we propose to extend the DFA described in [25] and [37] to any SPN-based block cipher, by an in-depth analysis of LS-Designs [18], a family of SPN-based block cipher for which the attack is the most effective. We first present physical attacks and LS-Designs before describing the theoretical DFA using several fault models. These attacks have been validated in practice on a software implementation of PRIDE [25], which follows a construction similar to LS-Designs. In order to validate the practical feasibility of our attack on a hardware implementation, we used electromagnetic pulses to inject faults during the execution of SCREAM running on an FPGA Xilinx Spartan-3E 1600E and we applied our DFA on the obtained corrupted results. SCREAM is the TAE [28] (Tweakable Authenticated Encryption) mode of the block cipher Scream which is an instantiation of LS-designs [20]. It should not be confused with the stream cipher Scream [23]. Then, we detail how to apply the DFA on other families of SPN-based block ciphers: the CUBE family, S-bP structures (i.e. SPN having a bit permutation as a linear layer) and AES-like structures. Finally we discuss countermeasures that can be implemented to thwart such attacks before concluding the paper.

2 Fault attacks against cryptographic primitives

Fault attacks consist in disturbing the behaviour of the circuit in order to alter the correct execution of the cipher. The faults are injected into the device by various means such as light pulses [36], laser [35], clock glitches [2], spikes on the voltage supply [9] or electromagnetic (EM) perturbations [15]. Some of those techniques, like the laser one, are invasive, requiring the "decapsulation" of the chip using mechanical or chemical means. Laser allows to target one bit in a given register if well manipulated [14]. However it is an expensive means of injection. Other techniques are not invasive such as glitches (power, clock, electromagnetic). Clock and voltage glitches disturb the whole component, and many injections usually have to be made before getting the specific faults required by an attack. EM glitches on the other hand allow to have relatively high spatial and temporal precisions using relatively low-cost equipment [15]. One of the objectives of fault attacks, especially when considering cryptographic primitives, is to perform Differential Fault Analysis (DFA). DFA, originally described in [7], [11], consists in retrieving a cryptographic key by comparing the correct ciphertexts with one or more faulty ones. DFA techniques have been described and successfully applied to most of the publicly known ciphers going from symmetric ciphers like the DES [7] or the AES [34] to asymmetric ones like RSA [11] or even more complex schemes like pairing-based systems [26]. In the particular field of lightweight cryptography, DFA have been proposed against ciphers like PRESENT [40], SPECK [39], TRIVIUM [29], PRINCE [37] or PRIDE [25]. DFA techniques are very efficient in retrieving the keys used during a cipher execution, usually requiring a few executions only. It is also quite complex to devise countermeasures

against such attacks because of the diversity of the possible injection methods and because the usually deployed countermeasures (like redundancy, error-correcting codes etc) have serious impacts on performances of the targeted cipher. Therefore, in our approach of analyzing the security of implementations of SPN-based block ciphers, we decided to first focus on their resistance against fault attacks as to identify possible attack paths and devise more efficient countermeasures in order to try to keep the performance characteristics of the original ciphers.

3 LS-Designs

In this paper, bits, bytes, rows and columns are numbered from left to right starting from 1. LS-designs are iterative SPN-based block ciphers composed of r rounds. They were introduced by Grosso & al. [18] in 2014. An LS-design takes as input an n-bit block and uses an n-bit key. The inner state of the cipher, as well as the plaintext, ciphertext, and key, are all represented as $\omega \times c$ bit arrays, with ω the number of rows and c the number of columns such that $n = \omega \cdot c$. The following notation is used for the intermediate values of the state within a round:

I_i	the input of the i-th round
X_i	the state after the key addition layer of the i-th round
Y_i	the state after the substitution layer of the i-th round
Z_i	the state after the linear layer of the i-th round
O_i	the output of the i-th round

A round $1 \leq i \leq r$ is composed of the following steps:

i. Add by an XOR the n-bit key K to the state: $X_i = I_i \oplus K$,
ii. Apply an ω-bit S-box $\mathcal{S}$ to each column of the state (i.e. apply the substitution layer $\mathcal{S}\mathsf{layer}$ to the state): $Y_i = \mathcal{S}\mathsf{layer}(X_i)$,
iii. Apply a bijective linear map $\mathcal{L}$, called L-box, operating on c-bit vectors, to each row (i.e. apply the linear layer $\mathcal{L}\mathsf{layer}$ to the state): $Z_i = \mathcal{L}\mathsf{layer}(Y_i)$,
iv. Add by an XOR an n-bit round constant $\mathcal{C}t_i$ to the state: $O_i = Z_i \oplus \mathcal{C}t_i$.

An LS-design is parametrized by the choice of r, ω, c, $\mathcal{S}$, $\mathcal{L}$ and the round constants $\mathcal{C}t_i$ for $1 \leq i \leq r$. In order to encrypt a plaintext, the cipher applies the r rounds as previously described, it then performs an XOR between the state and the key. Figure 1 shows the representation of the inner state of an LS-design with an example framed of the input of S-box and the input of L-box.

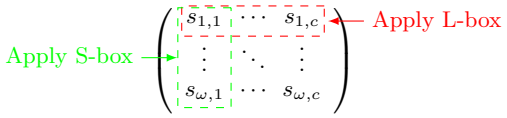

Figure 1: Inner state of an LS-design

Robin and Fantomas are two LS-designs proposed by Grosso & al. [18] in 2014 with $\omega = 8$ and $c = 16$. Scream and iScream are a modified version of Robin and Fantomas, referred to as Tweakable LS-Designs, also introduced by

Grosso & al. [20] with the same parameters. Those two ciphers are the core of two authenticated encryption schemes submitted to the Caesar competition [13]. It is worth noticing that these block ciphers have been recently broken in the sense that it has been shown that they have a large number of weak keys [38]. However, studying the security offered by the ciphers with respect to other attacks is still of interest. Indeed, the nonlinear-invariant attack can be prevented by changing the round constants, while such a change won't affect the resistance to DFA.

The main difference between Tweakable LS-Designs and LS-Designs lies in the addition of an input parameter, the Tweak, to thwart side-channel attacks if it is used and protected properly. It is a structural countermeasure which, as we shall see, also thwart our attack in this case. Finally, PRIDE, proposed by Albrecht & al. [3] in 2014, has a structure close to the one of an LS-Design with $\omega = 4$ and $c = 16$: it uses one additional key for pre- and post-whitening, uses several L-boxes within the linear layer and has no linear layer on the last round.

4 Applying DFA on LS-Designs

DFA against PRIDE, whose structure is similar to an LS-design, was first introduced in [25]. In this section, we propose a generalisation of fault attacks to any LS-design. First, we introduce the general principle to exploit fault injections. Then, we explain the different strategies depending on the fault model.

4.1 General principle

Despite some similarities, a DFA against a cryptographic algorithm is different from a classical differential analysis. Indeed, for the latter the differences must be injected on the input of the cipher while for a DFA it can be injected where the attacker wants. The DFA that we propose in this paper also differs from many existing DFA in the sense that, in our attack, the input and output differences of all active S-boxes are known to the attacker. Therefore, there is no need of guessing the input difference as in many other DFA. It consists in corrupting one L-box application (i.e corrupting a row of the state during the linear layer) in the penultimate round in order to obtain a known difference on the S-boxes inputs in the last round. More precisely, flipping the bit $1 \leq i \leq c$ of the row $1 \leq j \leq \omega$ gives a difference equal to $2^{\omega-j}$ (with a one only at position j) on the input of the i-th S-box in the last round. Moreover, from the knowledge of the correct and the faulty ciphertexts, we can compute the corresponding difference on the S-box output. Figure 2 shows for example the state difference obtained from a flip of the second row before the S-layer.

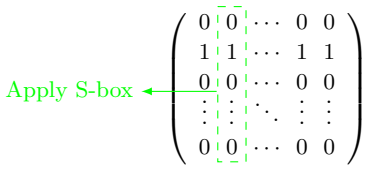

Figure 2: State difference obtained from a flip of the second row

In this case, we get a difference equal to $2^{\omega-1}$ on the input of each S-box. Thereby, if we denote ΔX_r (resp. ΔY_r) the difference input (resp. output) of the last substitution layer, we obtain a known differential $(\Delta X_r[i], \Delta Y_r[i])$ on the i-th S-box. Then, we exploit the difference distribution table of the S-box to reduce the number of remaining candidates for the secret key.

Indeed, using the notation introduced in Section 3, obtaining information on the key is possible from the following equations:

$$\Delta X_r = \mathcal{S}\mathsf{layer}^{-1}(\mathcal{L}\mathsf{layer}^{-1}(C \oplus K \oplus \mathcal{C}t_r)) \oplus \mathcal{S}\mathsf{layer}^{-1}(\mathcal{L}\mathsf{layer}^{-1}(C^* \oplus K \oplus \mathcal{C}t_r))$$
and
$$\Delta Y_r = \mathcal{L}\mathsf{layer}^{-1}(C \oplus K \oplus \mathcal{C}t_r) \oplus \mathcal{L}\mathsf{layer}^{-1}(C^* \oplus K \oplus \mathcal{C}t_r) = \mathcal{L}\mathsf{layer}^{-1}(\Delta C)$$

where C is the correct ciphertext and C^* the faulty one. We can use these equations for each ω-bit word $1 \leq i \leq c$:

$$x = \mathcal{L}\mathsf{layer}^{-1}(C \oplus K \oplus \mathcal{C}t_r)[i] \text{ and } y = \mathcal{L}\mathsf{layer}^{-1}(C^* \oplus K \oplus \mathcal{C}t_r)[i]$$
satisfy
$$x \oplus y = \Delta Y_r[i] = \mathcal{L}\mathsf{layer}^{-1}(\Delta C)[i] \text{ and } \mathcal{S}^{-1}(x) \oplus \mathcal{S}^{-1}(y) = \Delta X_r[i] = 2^{\omega-j}.$$

From the knowledge of a nonzero input difference $x \oplus y$ and of an output difference $\mathcal{S}^{-1}(x) \oplus \mathcal{S}^{-1}(y)$ for the inverse S-box $\mathcal{S}^{-1}$, we reduce the number of possible values for the input x. Moreover, from Proposition 1, we are able to easily find pairs of differentials for the S-box which are simultaneously satisfied for a single element. The proof to this proposition is given in Appendix A.

Proposition 1 *Let $\mathcal{S}$ be an n-bit S-box. Let (a_1, b_1) and (a_2, b_2) be two differentials with $a_1 \neq a_2$ such that the system of two equations*

$$\mathcal{S}(x \oplus a_1) \oplus \mathcal{S}(x) = b_1 \tag{1}$$

$$\mathcal{S}(x \oplus a_2) \oplus \mathcal{S}(x) = b_2 \tag{2}$$

has at least two solutions. Then, each of the three equations (1), (2) and

$$\mathcal{S}(x \oplus a_1 \oplus a_2) \oplus \mathcal{S}(x) = b_1 \oplus b_2$$

has at least four solutions.

In other words, if we can find two differentials (a_1, b_1) and (a_2, b_2) such that one out of the three entries in the difference distribution table (a_1, b_1), (a_2, b_2) and $(a_1 \oplus a_2, b_1 \oplus b_2)$ is equal to 2, then we can guarantee that the input satisfying these two differentials simultaneously is unique. Note that if one of the three equations has no solution then the system of two equations (1), (2) has no solution.

4.2 Ideal fault model

The ideal fault model consists simply in a first step in finding two output differences $b_1 = \Delta X_r^1 = 2^{\omega-i_1}$ and $b_2 = \Delta X_r^2 = 2^{\omega-i_2}$ with $1 \leq i_1 < i_2 \leq \omega$ such that (a_1, b_1) and (a_2, b_2) are simultaneously satisfied for a single element

for all a_1 and a_2. After, it is sufficient to flip the row i_1 then the row i_2 of the state during the penultimate linear layer with two successive fault injections in order to retrieve the complete secret key. Table 1 gives as example the pairs of differentials which are simultaneously satisfied for a single element in the case of the S-boxes involved in some LS-designs.

Table 1: Exploitable differential pairs

Cipher	Pair
PRIDE	(a_1,0x1), (a_2,0x8)
Robin	(a_1,0x01), (a_2,0x40)
Fantomas	(a_1,0x01), (a_2,0x80)
Scream	(a_1,0x01), (a_2,0x02)
iScream	(a_1,0x02), (a_2,0x80)

Note: Applying DFA on Scream and iScream is possible only if the attacker can execute encryption (or decryption) twice consecutively with the same tweak.

4.3 Random fault model

We call random fault model one where we have one chance out of two to flip each bit of a desired word. It is close to what is obtained in practice with electromagnetic pulses where it is possible to target a precise word (more precisely a specific instruction) but the injected faults follow a random distribution.

In order to achieve the attack, we must flip all the bits of two c-bit words in the ideal fault model used in the preceding part. However, we can see that flipping one bit provides an active S-box, it is therefore enough to flip all the bits of the desired c-bit words non simultaneously using as many faults as necessary. Accordingly, as in the ideal model, it consists first in finding two output differences $b_1 = \Delta X_r^1 = 2^{\omega - i_1}$ and $b_2 = \Delta X_r^2 = 2^{\omega - i_2}$ with $1 \leq i_1 < i_2 \leq \omega$ such that (a_1, b_1) and (a_2, b_2) are simultaneously satisfied for a single element for all a_1 and a_2. Let A_1 (resp. A_2) be the average number of remaining candidates for a key byte from an active S-box obtained from a fault on row i_1 (resp. i_2). Let m_1 (resp. m_2) denote the number of obtained faults on row i_1 (resp. i_2).

Then, the number of remaining candidates for the key from the faults and from the knowledge of ΔY_r for each fault is

$$N = \left(\frac{2^\omega}{2^{m_1+m_2}} + \sum_{i=1}^{m_1} \frac{A_1}{2^{i+m_2}} + \sum_{i=1}^{m_2} \frac{A_2}{2^{i+m_1}} + \left(\sum_{i=1}^{m_1} \frac{1}{2^i} \right) \left(\sum_{i=1}^{m_2} \frac{1}{2^i} \right) \right)^c .$$

Indeed, when m faults have been injected on one word, the probability to obtain no difference on a byte is equal to $1/2^m$ and the probability to obtain at least one is equal to $\sum_{i=1}^{m} 1/2^i = 1 - 1/2^m = (2^m - 1)/2^m$. Moreover, if we get no difference with all the faults (on the first and on the second word) then we still have 2^ω candidates for the corresponding byte. On the other hand, if we get only one difference we obtain A_1 or A_2 candidates. Finally, if we get two differences

we retrieve the correct value. We then deduce that

$$N = \left(\frac{2^\omega + A_1(2^{m_1} - 1) + A_2(2^{m_2} - 1) + (2^{m_1} - 1)(2^{m_2} - 1)}{2^{m_1+m_2}} \right)^c$$
$$= \left(\frac{2^\omega - A_1 - A_2 + 1}{2^{m_1+m_2}} + \frac{A_1 - 1}{2^{m_2}} + \frac{A_2 - 1}{2^{m_1}} + 1 \right)^c .$$

4.4 Properties that make the attack effective

Our attack mainly exploits the following 2 properties of the building-blocks of most SPN-based block ciphers. It is worth noticing that these two properties come from the fact that the building-blocks of the cipher have been designed for maximizing the resistance of the cipher against differential and linear cryptanalysis.

The design of the linear layer. Indeed, the motivation for LS-Designs, as well as the more general interleaved construction [3], is to guarantee a large number of active S-boxes in any differential or linear attack (see e.g. Theorem 1 in [3] and Page 23 in [18]). Indeed, in LS-Designs, flipping all bits of any row at the input of the linear layer activates all S-boxes in the next round. Indeed, by construction, the c bits of any row go to different S-boxes. It follows that flipping one row of the penultimate round allows the attacker to recover information on the whole subkey used in the last round. The same situation occurs in the interleaved construction. In other words, the optimal diffusion offered by those constructions enables the attacker to recover the whole last-round subkey. This would not be the case if the linear layer was weaker with respect to the classical diffusion criteria.

The differential properties of the S-box, which avoids the existence of differentials of high probability over a large number of rounds. The counterpart of this property required for resistance against classical differential cryptanalysis is that the number of inputs which satisfy two valid differentials simultaneously is usually reduced to a single element. In the context of a DFA, this property enables the attacker to drastically reduce the number of key candidates. In many cases, two faults are enough to obtain a single candidate for the secret key.

5 Practical implementation of the DFA on SCREAM

5.1 The TAE mode SCREAM

The TAE (Tweakable Authenticated Encryption) mode is a mode of operation introduced by Liskov & al. [27,28]. In order to encrypt a message M, the TAE mode splits it into m blocks of size n such that the last block is padded if necessary, and in this case it is added by an XOR to its output. Then, it applies on each block a tweakable block cipher

$$E : \{0,1\}^k \times \{0,1\}^t \times \{0,1\}^n \to \{0,1\}^n$$

using a key K of size k and a tweak T of size t. It uses the same key K to encrypt each block and different tweaks produced from the same nonce N (the recommended size for the nonce in SCREAM is 11 bytes). Finally, a tag is produced from the checksum of all blocks.

SCREAM optionally proposes to authenticate blocks of associated data with the message. It uses the tweakable block cipher Scream and the tweaks are produced from the same nonce concatenated with a block counter. Scream is an iterative block cipher composed of N_s steps, each of them made of N_r rounds, denoted Tweakable LS-Design and introduced by Grosso & al. [20] in 2014. The recommended parameters for a related-key security are $N_s = 12$ and $N_r = 2$. It takes as inputs a 128-bit block, a 128-bit key K and a 128-bit tweak $T = t_0||t_1$. The tweak is used as a "lightweight key schedule": the output of the step s is added by an XOR to a subkey equal to

$$\begin{array}{ll} K \oplus (t_0||t_1) & \text{if } s = 3i, \\ K \oplus (t_0 \oplus t_1||t_0) & \text{if } s = 3i+1, \\ K \oplus (t_1||t_0 \oplus t_1) & \text{if } s = 3i+2. \end{array}$$

Each round is composed of the following steps: it applies a nonlinear layer composed of 8-bit S-boxes, then it adds by an XOR a round constant and finally it applies a 16-bit L-boxes layer. Specifications on these components are given in Appendix of [20]. In order to encrypt a plaintext P, Scream adds by an XOR the first subkey to P (with $s = 0$), then it applies the N_s steps described before. The tweak, or the nonce in the case of the TAE mode, is a protocol-level countermeasure against side-channel analysis added directly into the design. It must be changed at each execution in order to be effective. In this case, we will see it also protects against our DFA. However, attacking a Tweakable LS-Design using a fixed tweak is equivalent to attacking an LS-Design, what interests us in our case to validate our DFA.

5.2 The DFA on SCREAM

Briefly, we recall the general principle of our attack on SCREAM. Firstly, we consider that the same nonce is used at each execution, i.e. each block uses the same tweak at each encryption. As mentioned, that provides a structure equivalent to an LS-Design. Then, flipping one row before the S-box layer in the last round during one execution of Scream allows to obtain known differentials on each S-box. The best case is to flip, from two faults, the last and the penultimate row since in this case the number of inputs which satisfy the obtained differentials (a_1,0x01) and (a_2,0x02) simultaneously is reduced to a single element. It can be verified by testing all intersections of the obtained sets from each possible pair of differentials. Thereby, an attacker can use these differentials for retrieving the key from the following equations, for each byte $1 \leq i \leq 16$:

$$x = \mathcal{L}\mathsf{layer}^{-1}(C \oplus K \oplus T)[i] \oplus \mathcal{C}t_{23}[i] \text{ and } y = \mathcal{L}\mathsf{layer}^{-1}(C^* \oplus K \oplus T)[i] \oplus \mathcal{C}t_{23}[i]$$
$$\text{satisfy}$$
$$x \oplus y = a_1 \text{ (resp. } a_2) \text{ and } \mathcal{S}^{-1}(x) \oplus \mathcal{S}^{-1}(y) = \text{0x01 (resp. 0x02)}.$$

Finally, in case of the ideal fault model, an attacker can retrieve the complete secret key from two faults only (flip two complete rows) since she knows C, T and $\mathcal{C}t_{23} = 50577$ (defined in [20]). In case of the random fault model, any differential (a_1,0x01) allows to obtain $A_1 \approx 2.286$ candidates for a key byte and any differential (a_2,0x02) allows to obtain $A_2 \approx 2.639$ candidates. Moreover, the inner state of Scream is represented as $\omega \times c$ bit arrays, with $\omega = 8$ the number of rows and $c = 16$ the number of columns. Therefore, the average number of remaining candidates for the key from m_1 (resp. m_2) random faults on the last (resp. penultimate) row is approximately:

$$\left(\frac{252.075}{2^{m_1+m_2}} + \frac{1.286}{2^{m_2}} + \frac{1.639}{2^{m_1}} + 1\right)^{16}$$

Table 2 gives the average number of remaining candidates for some values of m_1 and m_2. We note that 6 faults or more are enough to retrieve the key since in this case the attacker obtain less than 2^{40} remaining candidates.

Table 2: Number of remaining candidates for K

		m_1				
		1	2	3	4	5
m_2	1	$2^{96.5}$	$2^{81.2}$	$2^{66.4}$	$2^{52.6}$	$2^{40.6}$
	2	$2^{81.1}$	$2^{66.1}$	$2^{51.8}$	$2^{39.1}$	$2^{28.4}$
	3	$2^{66.2}$	$2^{51.7}$	$2^{38.5}$	$2^{27.2}$	$2^{18.5}$
	4	$2^{52.3}$	$2^{38.8}$	$2^{27.1}$	$2^{17.9}$	$2^{11.3}$
	5	$2^{39.9}$	$2^{27.9}$	$2^{18.2}$	$2^{11.2}$	$2^{6.7}$

5.3 Practical implementation of the DFA

In order to test the feasibility of our attack on a hardware implementation, we have implemented and run the 128-bit reference VHDL code of SCREAM, given in [19], on an FPGA Xilinx Spartan-3E 1600E manufactured with advanced 90 nm process technology using a frequency of 50 MHz. The FPGA die was composed of components CRYPTO, UART and FSM. The CRYPTO component contained the whole reference code of SCREAM, the UART allowed us to send data from the computer to the chip and the FSM allowed to define all the internal states. The input parameters used at each encryption was:

i. Nonce (11 bytes): 0xe9e6f9281b86c8470ba120,
ii. Key: 0x2ff6963dd72462ab67d5da22c0e264ae,
iii. Associated data (2 blocks): 0x5c0e6a47bc146679d2d64aca577463679782953401 57eb9d2581bfbb14a0cb39,
iv. Data (3 blocks): 0x6b36f33ff882e432861448a61183583b0df1f908593481535b6eb bc6abfc07ae22cd50a331678301fd8535690335dcbe.

The correct ciphertext was 0xc9018ef2804f85e0de4d6519593a3e5ed83c22bdc8b2db 2229e6801071cdea6785856feac83bbe335c6bcb2f5f6d81a6 and the tag was 0x84670e f3aaba9ee5d7358858c65c41ed. In practice, an attacker can target any block of

data to carry out the attack, she must just target the last linear layer of the execution of the running Scream. In our case, we injected all the faults during the last Scream execution. Therefore, we will give only the value of the last block for each fault, the correct value without fault being 0x85856feac83bbe335c6bcb2f5f6d81a6. First we performed a simple electromagnetic analysis of one SCREAM execution to identify the last round. We needed actually to only know the total duration of the encryptions in order to deduce the temporal position of the last round - which is always feasible in practice, even if there is an added noise. Figure 3 shows the obtained curves from the simple electromagnetic analysis.

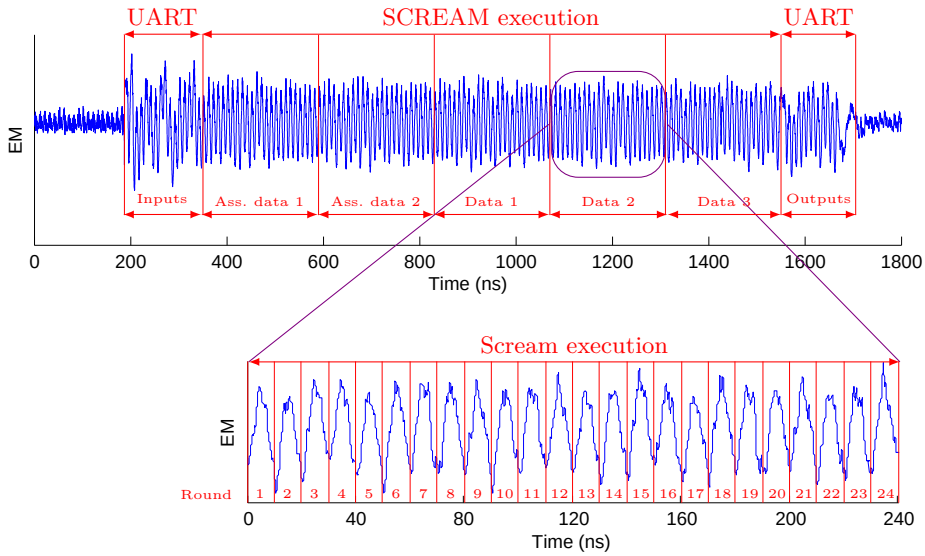

Figure 3: Simple electromagnetic analysis of SCREAM

So as to conduct the attack, we used electromagnetic pulses because with this approach we did not need to decapsulate the chip and we were able to inject faults at precise enough instants. The set-up we used is quite similar to the one described in [15]. In our case, the duration of the full encryption was approximately 1200ns, i.e. 240ns for one Scream encryption and 10ns for each round. A pulse duration, from our pulses generator, could be 6ns at minimum, which is almost half of a round. However, some signals are only used by the linear layer and the pulse can affect only few of them, which allowed us to obtain the desired fault model. First, we have done a cartography of the obtained faults on the full chip of size 19×19mm. We injected pulses on 100 positions distributed on a 10×10 grid. On each, we tested 11 different temporal positions, 4 different voltages and we injected 2 pulses, i.e. a total of 88 shots by spatial position. On the 8800 total injections, we obtained 465 faults of which at most 88 to one spatial position. Figure 4 shows the faults distribution on the chip. Then, we targeted the sensitive area of the chip - which probably corresponds to the FPGA die - and we injected a total of 69250 pulses. We obtained a total of 2482 faults, among which 937 were different. For each fault, we calculated the value of the

difference output on the last substitution layer and we verified if each byte could have been obtained by the same difference in input equal to 2^j with $0 \leq j \leq 7$.

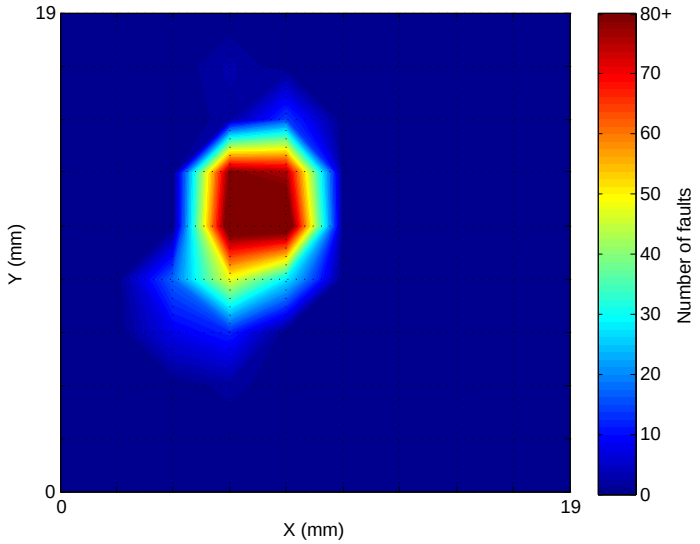

Figure 4: Cartography of the obtained faults on the full chip

A total of 36 different faults complied with this property. The obtained faults as well as our knowledge about the differences values around the last substitution layer are given in Appendix B. Finally, we obtained 6144 $\approx 2^{12,6}$ candidates for $\mathcal{L}\mathsf{layer}^{-1}(C \oplus K \oplus T) \oplus \mathcal{C}t_{23}$ and we retrieve K by testing all from the knowledge of C, T and $\mathcal{C}t_{23}$. The obtained number of remaining candidates does not correspond to theoretical analysis because EM pulses not allow to target a chosen row.

6 Application on other SPN-based block ciphers

6.1 Application on the CUBE family

A cipher belonging to the CUBE family (called CUBE) is an iterative SPN-based block cipher composed of r rounds whose concept was introduced by Berger & al. [6] in 2015. It takes as input an n-bit block and uses an n-bit or a $2n$-bit key with a key schedule defined in [6]. The inner state of the cipher, as well as the plaintext, ciphertext, and key, are all represented as a $\omega \times \omega \times \omega$ cube. The cube is filled beginning with its least significant bit at position (1, 1, 1) according the reference (X, Y, Z). A round $1 \leq i \leq r$ is then composed of the following steps:

i. Add by an XOR an n-bit subkey SK_i to the state,
ii. Apply an ω-bit S-box $\mathcal{S}$ to each row of X,
iii. Apply a quasi-involutive Feistel-MDS transformation [33], denoted M, on ω words of size ω bits, for each plane (i, Y, Z), $1 \leq i \leq \omega$,
iv. Rotate the axes (X, Y, Z) as (Z, X, Y).

An instance is parametrized by the choice of r, ω, $\mathcal{S}$ and M. In order to encrypt a plaintext, the cipher applies the r rounds as previously described, it then performs an XOR between the state and a last subkey SK_r. Figure 5 shows the representation of the inner state of a CUBE instance with $\omega = 4$ illustrating an example of the input of $\mathcal{S}$ and the input of M.

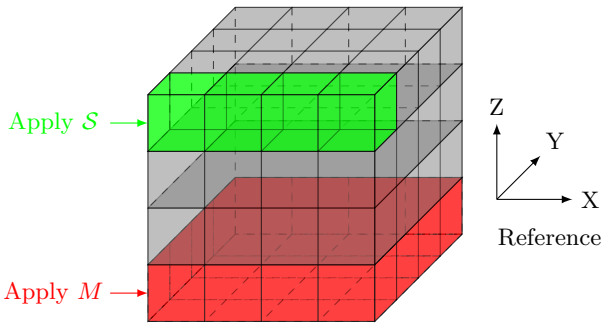

Figure 5: Inner state of a CUBE

CUBE proposed in [6] is an example with $\omega = 4$. The decryption process of PRESENT-80 is also an example with $\omega = 4$ but without the quasi-involutive Feistel-MDS transformation. The DFA consists in this case in flipping a $\omega \times \omega$ plane in X before the axes rotation, i.e. during the quasi-involutive Feistel-MDS transformation, in order to obtain known differences at the inputs of all the S-boxes on the last round. Indeed, flipping the plane $1 \leq k \leq \omega$ in X before the axes rotation allows to obtain differences equal to 1 only on the plane k in Z after it, i.e. differences equal to 2^{k-1} on each of the S-box inputs. Thereby, an attacker can run the DFA on the last round to retrieve the last subkey and repeat the attack on the previous rounds until she recovers enough key information-bits to retrieve the complete key from the key schedule.

6.2 Application on S-bP structures

An S-bP structure is an iterative SPN-based block cipher with a bit permutation layer. It takes as input an n-bit block, uses an n-bit key with a key schedule, a ω-bit S-box and a bitwise permutation layer which diffuses each S-box output to different S-boxes input. Such a cipher with r rounds is called an S-bP(n, ω, r) structure according to these parameters. A round consists of the following steps:

i. Add by an XOR the current n-bit subkey to the state,
ii. Divide the state into n/ω words and apply the ω-bit S-box to each word,
iii. Apply the bitwise permutation layer.

PRESENT-80 is an S-bP$(64, 4, 31)$ structure introduced by Bogdanov & al [10] in 2007. PRINTCIPHER is an S-bP$(32, 3, 48)$ structure proposed by Knudsen & al [24] in 2010. The DFA consists in this case in flipping the output of one S-box in the penultimate round to obtain known differences at input of several S-boxes

on the last round thanks to the design of the bitwise permutation layer. Figure 6 shows the diffusion of a difference obtained from a flip on the output of a 4-bit S-box before the permutation layer of an S-bP structure, which allows to obtain 4 differences equals to 0x8 at the input of the next substitution layer. It is the case for a flip of the first S-box output on a round of PRESENT-80 for example.

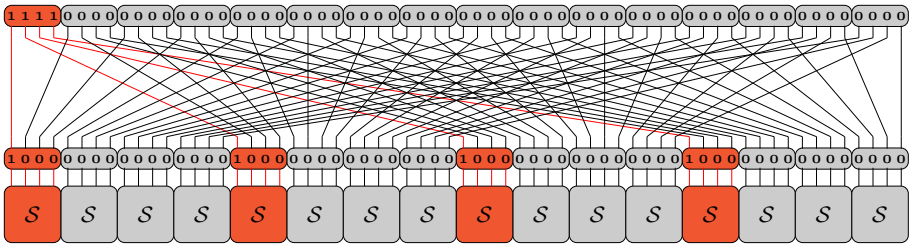

Figure 6: Propagation of a difference obtained by a flip of a nibble before the permutation layer of PRESENT-80

In case of PRESENT-80, the differentials (a_1,0x1) and (a_2,0x8) are simultaneously satisfied for a single element for all a_1 and a_2. Using this attack, the attacker can retrieve the complete key from the key schedule once she gets the last two subkeys from faults on the nibbles 0, 3, 4, 7, 8, 11, 12 then 15 before the permutation layer in the last two rounds. Note that the attack on the decryption needs only two faults to retrieve a subkey: on the first and the last 16-bit word.

6.3 Application on AES-like structures

An AES-like structure is an iterative SPN-based block cipher composed of r rounds. It takes as input an n-bit block and uses an n-bit key with a key schedule. The inner state of the cipher is represented as an $l \times c$ ω-bit array, with l the number of rows and c the number of columns. The size of the input and the key is $n = \omega \cdot l \cdot c$. A round is composed of the following steps:

i. Add by an XOR an n-bit round constant to the state,
ii. Add by an XOR the current n-bit subkey to the state,
iii. Apply an ω-bit S-box to each cell of the state,
iv. Apply shifts of the cells on each row of the state,
v. Apply a matrix multiplication transformation to each column of the state,

Midori64 is an AES-like structure, introduced by Banik & al [4] in 2015, with $\omega = 4$, $l = 4$ and $c = 4$. KLEIN-64 is another example, introduced by Gong & al [17] in 2012, with $\omega = 4$, $l = 8$ and $c = 2$. LED, proposed by Guo & al [22] in 2011, is also an example with $\omega = 4$, $l = 4$ and $c = 4$. The DFA consists in this case in flipping a row before the matrices application (step v), i.e. flipping a c-bit word of the state, in order to obtain known differences at the input of each S-box. Unlike other families of SPN-based block ciphers, the AES-like structures do not use a bitwise permutation (which is transparent in the case of LS-designs

but which is however exploitable). The bitwise permutation allows us to obtain differences equal to 0x1, 0x2, 0x4 or 0x8 at input of each S-box which allows a highly efficient exploitation of the obtained faults. In the case of AES-like structures, the differences obtained mainly depend on the matrices used, making these designs more resistant to our attack, although it remains relevant. For example, flip the first row then the third row of the state after the substitution layer of the AES-like structure LED allows to respectively obtain the following state differences before the substitution layer in the next round:

$$\begin{pmatrix} 0x9 & 0x9 & 0x9 & 0x9 \\ 0x1 & 0x1 & 0x1 & 0x1 \\ 0x3 & 0x3 & 0x3 & 0x3 \\ 0xd & 0xd & 0xd & 0xd \end{pmatrix} \text{ and } \begin{pmatrix} 0xd & 0xd & 0xd & 0xd \\ 0x6 & 0x6 & 0x6 & 0x6 \\ 0xc & 0xc & 0xc & 0xc \\ 0xa & 0xa & 0xa & 0xa \end{pmatrix}$$

Moreover, pairs of differentials $\{(a_1,0x9), (a_2,0xd)\}$, $\{(a_1,0x1), (a_2,0x6)\}$, $\{(a_1,0x3), (a_2,0xc)\}$ and $\{(a_1,0xd), (a_2,0xa)\}$ on the inverse S-box of LED guarantee that the input satisfying these two simultaneously is unique. However, if the attacker does not control the value of the fault, it is more difficult to exploit it because she cannot predict the previous state difference before the last substitution layer.

7 Countermeasures

An LS-design and more generally a block cipher is always used following a well-defined mode of operation. We will show that a number of such modes intrinsically protect against our attack. Then, we will present and briefly analyze possible countermeasures to thwart DFA: masking and the so-called Internal Redundancy Countermeasure (IRC) which we propose as a new kind of countermeasure.

7.1 Modes of operation

In order to encrypt data, a block cipher is always used with a mode of operation. It turns out that some well-known modes - standardized and already used in practice - thwart our DFA. Therefore, in this usage context, it is not necessary to add a countermeasure to protect the cipher. It is the case for the modes which use a random initialization vector, denoted IV, to encrypt data, as for example the OFB mode, which is illustrated in Figure 7.

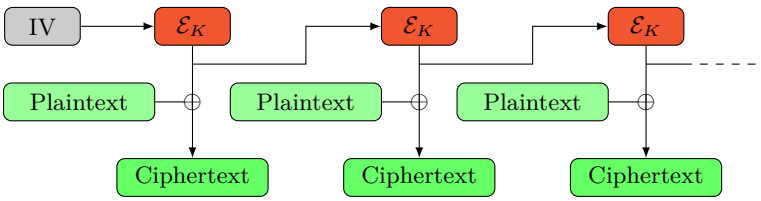

Figure 7: OFB mode encryption

This mode applies the cipher $\mathcal{E}_K$ directly on the IV and thus manipulate the plaintext only to add it to the obtained output. The IV changes at each

execution and cannot be controlled by the attacker to comply with the correct use of these modes of operation. Therefore, an attacker cannot by construction have two executions of the cipher with the same input and so she cannot apply a DFA since it is a necessary condition. It is also the case for the CTR mode which combines a unique nonce and a counter which is incremented at each execution of the algorithm, therefore guaranteeing the latter is never run twice on the same plaintext input. Some modes of operation like the CBC mode add the IV directly to the plaintext P and apply the cipher on the result. In this case, the IV must be unpredictable by the attacker in advance, otherwise the attacker can mount an adaptive plaintext attack by using two pairs of plaintext and IV, (IV_1, P_1) and (IV_2, P_2) such that $P_1 \oplus IV_1 = P_2 \oplus IV_2$.

Now, we will provide two countermeasures to thwart DFA when the mode of operation leaves the cipher unprotected against it, for example when used to derive a keyed one-way function for authentication.

7.2 Masking

Description: A countermeasure proposed by Guilley & al. in [21] is to add a random mask to the message to prevent two consecutive executions of the same plaintext. More precisely, in its original description, it consists in generating a random mask different at each execution, to XOR it to the plaintext and to return the corresponding ciphertext with the mask. However, in our case, the mask generator must be unpredictable, otherwise an attacker can choose an adaptive plaintext as described in the previous section.

Another technique - which we have already hinted at - is to use a tweakable cipher. The tweak can be considered like a mask which is added at each step and must be changed at each execution. This countermeasure, originally proposed for avoiding side-channel attacks, also protects against DFA since, once again, an attacker cannot obtain two executions on the same plaintext.

In order to guard only against DFA, we propose to use only one mask (different for each execution of the cipher) which must be added to the state SM in the middle of the encryption $\mathcal{E}_K$. More precisely, if $\mathcal{E}_K$ is composed of r rounds and takes an n-bit block as input, the countermeasure consists in computing

$$\mathcal{E}_K^{(1)}(\mathcal{E}_K^{(0)}(\text{Plaintext}) \oplus \text{RV})$$

where RV is an n-bit random value and $\mathcal{E}_K^{(0)}$ (resp. $\mathcal{E}_K^{(1)}$) corresponds to the first $\lceil r/2 \rceil$ rounds (resp. last $\lfloor r/2 \rfloor$ rounds) of the cipher. The decryption $\mathcal{D}_K = \mathcal{D}_K^{(1)} \circ \mathcal{D}_K^{(0)}$ must be synchronized with encryption (like for a tweakable cipher). In that respect, we can use the same process as a mode of operation which synchronizes the IV for encryption and decryption and which can therefore be expected to be already available in existing systems. Figure 8 illustrates the countermeasure with the introduced notation. Then, the mask generator can be public if we assume that the attacker does not have access to the encryption and decryption functions, both parametrized by the same key and mask. This is a necessary condition for each masking we have presented, otherwise the attacker can lead the DFA on the decryption since she knows the correct plaintext.

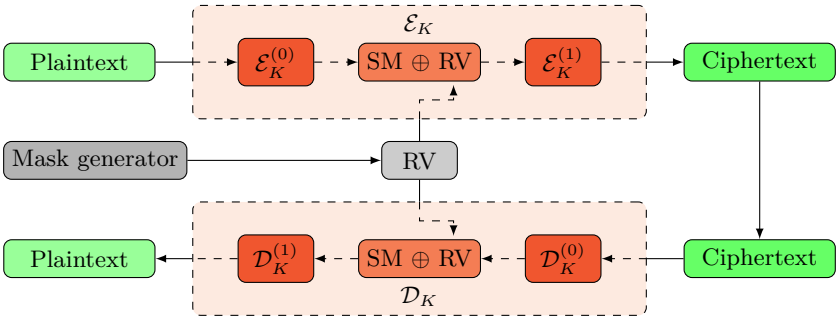

Figure 8: Masking countermeasure

Indeed, to mount a DFA on the encryption, an attacker must obtain a correct ciphertext $C = \mathcal{E}_K^{(1)}(\mathcal{E}_K^{(0)}(P_1) \oplus RV_1)$ and a faulty ciphertext $C^* = \mathcal{E}_K^{(1)}(\mathcal{E}_K^{(0)}(P_2) \oplus RV_2)$ such that the inputs of $\mathcal{E}_K^{(1)}$ are the same in both computations, i.e.

$$\mathcal{E}_K^{(0)}(P_1) \oplus RV_1 = \mathcal{E}_K^{(0)}(P_2) \oplus RV_2. \tag{3}$$

Similarly, if the attack is mounted against the decryption function, the inputs of $\mathcal{E}_K^{(0)}$ must be the same in both computations. There are two strategies for finding two pairs of inputs which satisfy (3). The first one consists in using a generic algorithm (without exploiting any specific property of the cipher). From the birthday paradox, this requires $2^{n/2}$ encryptions where n is the block size. In our case, the attacker has then to perform 2^{32} fault injections, which is infeasible in practice. A second strategy consists in exploiting some differential properties of $\mathcal{E}_K^{(0)}$. But this is again infeasible if $\mathcal{E}_K^{(0)}$ does not have any differential of probabilty higher than 2^{32}. Therefore, the mask generator can be a simple LFSR implemented in hardware which must not be modifiable by the attacker.

Cost: The cost depends on the choice of the random mask generation. A simple LFSR implemented in hardware has a low cost with respect to IoT constraints.

7.3 Internal Redundancy Countermeasure

Description: Recently, a countermeasure based on Intra-Instruction Redundancy [30] was proposed to thwart fault attacks. It uses a bit-sliced implementation of a given cipher applied to 15 blocks of data interleaved with 15 blocks of redundancy and 2 blocks of references in order to fit with a 32-bit architecture. The blocks of references are constant plaintexts for which the corresponding ciphertexts are known. Unfortunately, this countermeasure imposes to use - in most cases - a less efficient implementation of the cipher due to the Boolean circuit transformation overhead necessary for bit-slicing [32] and need to encrypt data blocks 15 by 15 from n encryption for an n-bit input. However, using reference blocks as part of a countermeasure is very effective against skip instruction. Thereby, we investigated the possibility to keep this property, also keep a spatial redundancy, while using a conventional (i.e. non-bitsliced) implementation of

a cipher applied on only one input block. Hence, we propose the Internal Redundancy Countermeasure (IRC) which exploits efficient 8-bit implementations - which is usually the preferred option for ciphers used in IoT devices, even more for LS-Designs - on a 32-bit architecture. IRC consists in using the original implementation with the same operations but from 32-bit instructions systematically operating as a whole on the 4 bytes of a 32-bit word.

Let $\mathcal{E}$ denote a cipher which takes as input a b-byte plaintext $P = P_1 \cdots P_b$, uses a b'-byte key $K = K_1 \cdots K_{b'}$ and produces a b-byte ciphertext $C = C_1 \cdots C_b$. IRC uses a b-byte reference plaintext $RP = RP_1 \cdots RP_b$, a b'-byte reference key $RK = RK_1 \cdots RK_{b'}$ and a b-byte reference ciphertext $RC = RC_1 \cdots RC_b$. Figure 9 shows one encryption protected by IRC.

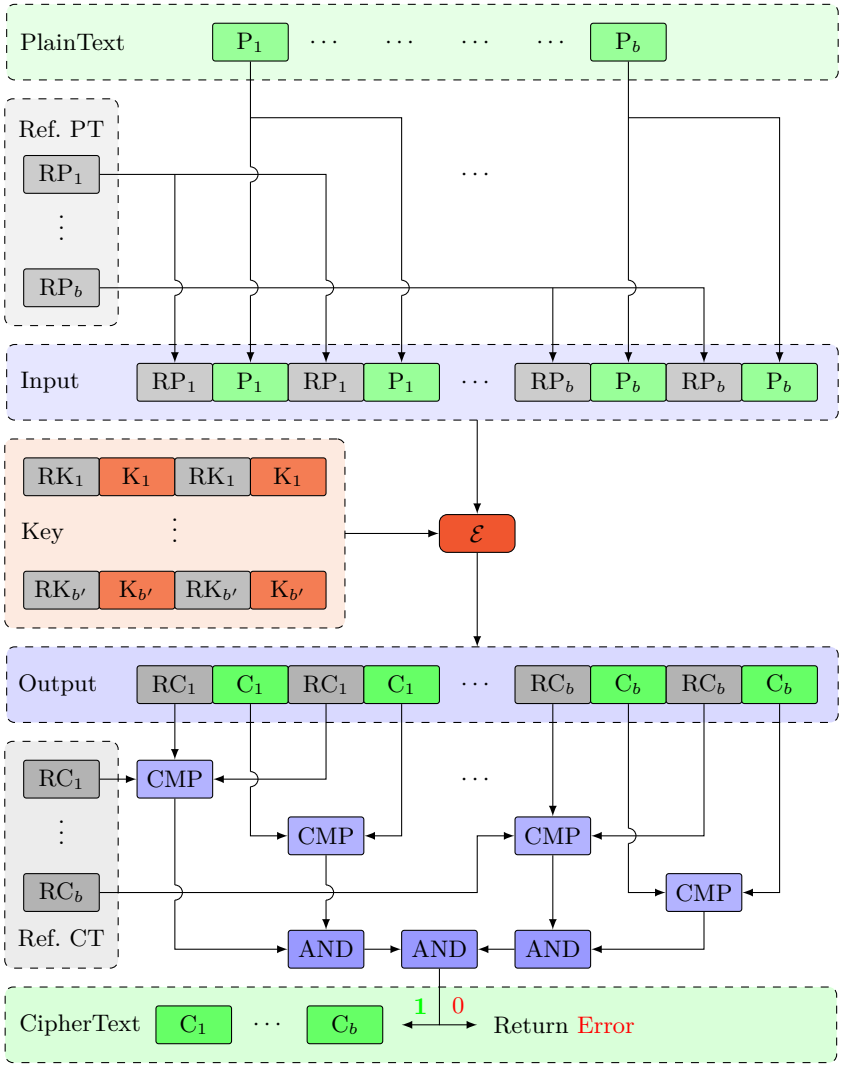

Figure 9: IRC process

Firstly, for each $i \in \{1, \cdots, b\}$, IRC stores in a 32-bit word the byte P_i concatenated with RP_i, P_i and RP_i. IRC also stores, for each $i \in \{1, \cdots, b'\}$, the byte K_i concatenated with RK_i, K_i and RK_i. Then, it applies the cipher by means of a single stream of 32-bit instructions to obtain, for each $i \in \{1, \cdots, b\}$, the byte C_i concatenated with RC_i, C_i and RC_i. Finally, it makes comparisons between redundant bytes and with the reference ciphertexts.

The resulting structure intrinsically protects against fault attacks. Indeed, if the fault is an instruction skip, which corresponds to a random fault for our DFA, the value of the reference ciphertext will be different, so the fault will be detected and the system trapped. Now, if the fault directly affects the value contained in a 32-bits word manipulated by the cipher, it must not affect the first and the third bytes and it must have the same impact on the second and the last bytes. This last case is extremely difficult to control in practice.

Cost: LS-Designs mainly use bitwise operators like logical AND, OR, exclusive OR, shift etc. and also nonlinear operators over $\mathbb{F}_2^8$ like addition, multiplication, modulo etc. operating on bytes. Thereby, IRC simply uses bitwise operators on 32-bit - with sometimes minor changes like the shift for which it must use one mask - and use masks to implement nonlinear operators using few additional 32-bit instructions systematically operating as a whole on the 4 bytes of a 32-bit word to ensure the unicity of the instruction stream. Finally, IRC can use SIMD instructions, depending on the targeted device, to replace some nonlinear operations. Therefore, we obtain performances close to those on an 8-bit architecture while having a structure that intrinsically protects against DFA.

8 Conclusion

In this paper we propose a general method for differential fault analysis on any block cipher based on LS-designs and other families of SPN with similar structures. Such an approach had already been used against a software implementation of PRIDE [25], whose design is close to the one of an LS-Design. But our generalisation has allowed us to successfully perform such an attack against a hardware implementation of SCREAM [20], using the TLS-Design Scream with a fixed tweak. Faults were injected using electromagnetic pulses, which constitutes a low-cost means of injection. We believe that the resistance against DFA is important for LS-Designs, which are expected to be largely deployed in low-resource connected devices. Finally, we propose some countermeasures to thwart such attacks while keeping the efficiency of the ciphers for IoT devices, especially the so-called Internal Redundancy Countermeasure which we propose as a new kind of countermeasure.

Acknowledgement. Benjamin Lac's research work is partly supported by the French DGA-MRIS scholarship.

References

1. Adomnicai, A., Lac, B., Canteaut, A., Fournier, J.J., Masson, L., Sirdey, R., Tria, A.: On the importance of considering physical attacks when implementing lightweight cryptography. In: NIST Lightweight Cryptography Workshop 2016. Gaithersburg, Maryland (October 2016)
2. Agoyan, M., Dutertre, J., Naccache, D., Robisson, B., Tria, A.: When clocks fail: On critical paths and clock faults. In: Gollmann, D., Lanet, J., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 182–193. Springer, Berlin, Germany, Passau, Germany (April 14-16, 2010)
3. Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçin, T.: Block ciphers - focus on the linear layer (feat. PRIDE). In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 57–76. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014)
4. Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: ASIACRYPT 2015 - Part II. LNCS, vol. 9453, pp. 411–436. Springer, Berlin, Germany (Dec 2015)
5. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: SIMON and SPECK: Block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/585 (2015), http://eprint.iacr.org/2015/585
6. Berger, T.P., Francq, J., Minier, M.: Cube cipher: A family of quasi-involutive block ciphers easy to mask. In: El Hajji, S., Nitaj, A., Carlet, C., Souidi, E.M. (eds.) Codes, Cryptology, and Information Security: First International Conference, C2SI 2015, Rabat, Morocco, May 26-28, 2015, Proceedings - In Honor of Thierry Berger. pp. 89–105. Springer International Publishing, Cham (2015), http://dx.doi.org/10.1007/978-3-319-18681-8_8
7. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 513–525. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 17–21, 1997)
8. Bilgin, B., Bogdanov, A., Knezevic, M., Mendel, F., Wang, Q.: Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In: Bertoni, G., Coron, J. (eds.) Cryptographic Hardware and Embedded Systems, CHES 2013. Lecture notes in computer science, vol. 8086, pp. 142–158. Springer, Heidelberg, Germany (2013), http://doc.utwente.nl/89342/
9. Blömer, J., Seifert, J.P.: Fault based cryptanalysis of the advanced encryption standard (AES). In: Wright, R. (ed.) FC 2003. LNCS, vol. 2742, pp. 162–181. Springer, Berlin, Germany, Guadeloupe, French West Indies (Jan 27–30, 2003)
10. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Berlin, Germany, Vienna, Austria (Sep 10–13, 2007)
11. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults (extended abstract). In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 37–51. Springer, Berlin, Germany, Konstanz, Germany (May 11–15, 1997)
12. Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knežević, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., Rombouts, P., Thomsen, S.S., Yalçin, T.: PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 208–225. Springer, Berlin, Germany, Beijing, China (Dec 2–6, 2012)

13. CAESAR: Competition for Authenticated Encryption (2014), https://competitions.cr.yp.to/caesar.html
14. Courbon, F., Loubet-Moundi, P., Fournier, J.J., Tria, A.: Adjusting laser injections for fully controlled faults. In: Prouff, E. (ed.) Constructive Side-Channel Analysis and Secure Design. pp. 229–242. Lecture Notes in Computer Science, Springer International Publishing (2014)
15. Dehbaoui, A., Dutertre, J., Robisson, B., Tria, A.: Electromagnetic transient faults injection on a hardware and a software implementations of AES. In: Bertoni, G., Gierlichs, B. (eds.) FDTC 2012. pp. 7–15. IEEE Computer Society, Leuven, Belgium (September 9, 2012)
16. Gérard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.: Block ciphers that are easier to mask: How far can we go? Cryptology ePrint Archive, Report 2013/369 (2013), http://eprint.iacr.org/2013/369
17. Gong, Z., Nikova, S., Law, Y.W.: Klein: A new family of lightweight block ciphers. In: Juels, A., Paar, C. (eds.) RFIDSec 2011. pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-25286-0_1
18. Grosso, V., Leurent, G., Standaert, F.X., Varici, K.: LS-designs: Bitslice encryption for efficient masked software implementations. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 18–37. Springer, Berlin, Germany, London, UK (Mar 3–5, 2015)
19. Grosso, V., Leurent, G., Standaert, F.X., Varıcı, K., Journault, A., Durvaux, F., Gaspar, L., Kerckhof, S.: Implementations of the SCREAM authenticated encryption algorithm, https://perso.uclouvain.be/fstandae/SCREAM
20. Grosso, V., Leurent, G., Standaert, F.X., Varıcı, K., Journault, A., Durvaux, F., Gaspar, L., Kerckhof, S.: SCREAM, Side-Channel Resistant Authenticated Encryption with Masking (August 2015), https://competitions.cr.yp.to/round2/screamv3.pdf, submission to the CAESAR competition
21. Guilley, S., Sauvage, L., Danger, J., Selmane, N.: Fault injection resilience. In: Breveglieri, L., Joye, M., Koren, I., Naccache, D., Verbauwhede, I. (eds.) FDTC 2010. pp. 51–65. IEEE Computer Society, Santa Barbara, California, USA (August 21, 2010), http://dx.doi.org/10.1109/FDTC.2010.15
22. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.: The led block cipher. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917. Springer, Berlin, Germany, Nara, Japan (Sep 28 – Oct 1, 2011)
23. Halevi, S., Coppersmith, D., Jutla, C.: Scream: A Software-Efficient Stream Cipher, pp. 195–209. Springer Berlin Heidelberg, Berlin, Heidelberg (2002), http://dx.doi.org/10.1007/3-540-45661-9_15
24. Knudsen, L.R., Leander, G., Poschmann, A., Robshaw, M.J.B.: PRINTcipher: A block cipher for IC-printing. In: Mangard, S., Standaert, F.X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 16–32. Springer, Berlin, Germany, Santa Barbara, California, USA (Aug 17–20, 2010)
25. Lac, B., Beunardeau, M., Canteaut, A., Fournier, J.J., Sirdey, R.: A first DFA on PRIDE: from theory to practice. In: Proc. 11th International Conference on Risks and Security of Internet and Systems. Springer, Roscoff, France (September 2016)
26. Lashermes, R., Fournier, J., Goubin, L.: Inverting the final exponentiation of Tate pairings on ordinary elliptic curves using faults. In: Bertoni, G., Coron, J.S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 365–382. Springer, Berlin, Germany, Santa Barbara, California, US (Aug 20–23, 2013)
27. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. In: Yung, M. (ed.) CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 31–46. Springer (2002)

28. Liskov, M., Rivest, R.L., Wagner, D.: Tweakable block ciphers. Journal of Cryptology 24(3), 588–613 (2011), http://dx.doi.org/10.1007/s00145-010-9073-y
29. Mohamed, M.S.E., Bulygin, S., Buchmann, J.A.: Using SAT solving to improve differential fault analysis of Trivium. In: Kim, T., Adeli, H., Robles, R.J., Balitanas, M.O. (eds.) ISA 2011. Communications in Computer and Information Science, vol. 200, pp. 62–71. Springer, Berlin, Germany, Brno, Czech Republic (August 15-17, 2011)
30. Patrick, C., Yuce, B., Ghalaty, N., Schaumont, P.: Lightweight fault attack resistance in software using intra-instruction redundancy. In: 23rd Conference on Selected Areas in Cryptography (2016)
31. Piret, G., Roche, T., Carlet, C.: Picaro – a block cipher allowing efficient higher-order side-channel resistance. In: Bao, F., Samarati, P., Zhou, J. (eds.) Applied Cryptography and Network Security: 10th International Conference, ACNS 2012, Singapore, June 26-29, 2012. Proceedings. pp. 311–328. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-31284-7_19
32. Pornin, T.: Implantation et optimisation des primitives cryptographiques. Ph.D. thesis, Université Paris 7 (2001)
33. Sajadieh, M., Dakhilalian, M., Mala, H., Sepehrdad, P.: Recursive diffusion layers for block ciphers and hash functions. In: Canteaut, A. (ed.) Fast Software Encryption: 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. pp. 385–401. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-34047-5_22
34. Sakiyama, K., Li, Y., Iwamoto, M., Ohta, K.: Information-theoretic approach to optimal differential fault analysis. IEEE Transactions on Information Forensics and Security 7(1), 109–120 (2012)
35. Skorobogatov, S.: Semi-invasive attacks - A new approach to hardware security analysis. Technical Report 630, University of Cambridge (April 2005)
36. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski Jr., B.S., Koç, Çetin Kaya., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Berlin, Germany, Redwood Shores, California, USA (Aug 13–15, 2003)
37. Song, L., Hu, L.: Differential fault attack on the PRINCE block cipher. Cryptology ePrint Archive, Report 2013/043 (2013), http://eprint.iacr.org/2013/043
38. Todo, Y., Leander, G., Sasaki, Y.: Nonlinear invariant attack - practical attack on full scream, iscream, and midori64. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016 - Part II. Lecture Notes in Computer Science, vol. 10032, pp. 3–33 (2016)
39. Tupsamudre, H., Bisht, S., Mukhopadhyay, D.: Differential fault analysis on the families of SIMON and SPECK ciphers. Cryptology ePrint Archive, Report 2014/267 (2014), http://eprint.iacr.org/2014/267
40. Zhao, X., Wang, T., Guo, S.: Improved side channel cube attacks on PRESENT. Cryptology ePrint Archive, Report 2011/165 (2011), http://eprint.iacr.org/2011/165

A Differential properties of S-boxes

A.1 Proof of Proposition 1

Proof (of Proposition 1). Let $\mathcal{D}(a,b)$ denote the set of solutions of the equation

$$\mathcal{S}(x \oplus a) \oplus \mathcal{S}(x) = b.$$

Let us consider (a_1, b_1) and (a_2, b_2) be two differentials with $a_1 \neq a_2$ such that

$$\#\mathcal{D}(a_1, b_1) \cap \mathcal{D}(a_2, b_2) \geq 2.$$

It is clear that any element x in $\mathcal{D}(a_1, b_1) \cap \mathcal{D}(a_2, b_2)$ is a solution of

$$\mathcal{S}(x \oplus a_2) \oplus \mathcal{S}(x \oplus a_1) = b_1 \oplus b_2,$$

i.e., $x \oplus a_1 \in \mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$ and $x \oplus a_2 \in \mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$.
Let $\{x, x \oplus a_4\} \subseteq \mathcal{D}(a_1, b_1) \cap \mathcal{D}(a_2, b_2)$ for some $a_4 \neq 0$. Then

$$\{x, x \oplus a_1, x \oplus a_4, x \oplus a_1 \oplus a_4\} \subseteq \mathcal{D}(a_1, b_1),$$

$$\{x, x \oplus a_2, x \oplus a_4, x \oplus a_2 \oplus a_4\} \subseteq \mathcal{D}(a_2, b_2),$$

$$\{x \oplus a_1, x \oplus a_2, x \oplus a_1 \oplus a_4, x \oplus a_2 \oplus a_4\} \subseteq \mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2).$$

Since $a_1 \neq a_2$, we just must prove that if $a_4 = a_1$ or $a_4 = a_2$ or $a_4 = a_1 \oplus a_2$ then $\mathcal{D}(a_1, b_1)$, $\mathcal{D}(a_2, b_2)$ and $\mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$ have each at least 4 elements.
If $a_4 = a_1$ then $x \oplus a_1 \in \mathcal{D}(a_2, b_2)$ imply

$$\mathcal{S}(x \oplus a_1 \oplus a_2) \oplus \mathcal{S}(x \oplus a_1) = b_2 = \mathcal{S}(x \oplus a_2) \oplus \mathcal{S}(x)$$

implying that

$$\mathcal{S}(x \oplus a_1) \oplus \mathcal{S}(x) = \mathcal{S}(x \oplus a_2 \oplus a_1) \oplus \mathcal{S}(x \oplus a_2).$$

Thus $x \oplus a_2 \in \mathcal{D}(a_1, b_1)$ and $\mathcal{D}(a_1, b_1)$, $\mathcal{D}(a_2, b_2)$ and $\mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$ contain $\{x, x \oplus a_1, x \oplus a_2, x \oplus a_1 \oplus a_2\}$. It's identical if $a_4 = a_2$ following the same reasoning. Now, $a_4 = a_1 \oplus a_2$ implies that $x \oplus a_2 \oplus a_4 = x \oplus a_1$ belongs to $\mathcal{D}(a_2, b_2)$, i.e., $x \oplus a_1 \in \mathcal{D}(a_1, b_1) \cap \mathcal{D}(a_2, b_2)$. Therefore, $x \oplus a_1$, $x \oplus a_2$, x and $x \oplus a_1 \oplus a_2$ all belong to $\mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$ and $\#\mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2) \geq 4$. □

B Different exploitable faults obtained on SCREAM

Table 3 provides the 36 different exploitable faults obtained on the reference hardware implementation of SCREAM as well as our knowledge about the differences values around the last substitution layer for each fault, i.e. the value of the output difference ΔY_{24} and the possible values for each input byte difference, denoted ΔIn (which must be the same for all bytes). Among these faults, only 7 displayed in red gave as much information as all faults.

Table 3: Exploitable faults obtained on SCREAM

No	Value of the last block	Value of ΔY_{24}	Value of ΔIn
1	0x04eb0f2430df5f9301047fae10109f6d	0x003347ca19002a00d95d000000548a00	0x01
2	0x907766fd6347e9904999306543889d43	0x003300ca000000d5d900fd0000000000	0x01
3	0xc0414fd2280187f719af457254f68a3d	0x000000ca00002a00d900fd002a0000c2	0x01
4	0xc0416fea2801fbf719af2b151aa9c462	0x00000000000000000d900fd0000000000	0x01
5	0xc041e0232801280619afea8354f68a3d	0x000000ca00000000d900fd002a0000c2	0x01
6	0xc098d36128d80ce3760cea8308179d43	0x000047ca000000d5d95dfd002a000000	0x01
7	0xdb313ac52801e99002df306554f68a3d	0x003300ca00000000d900fd00000000c2	0x01
8	0xdb3154a2280187f702df5e0254f68a3d	0x003300ca00002a00d900fd002a0000c2	0x01
9	0xdb3166fd2801e99002df306508ced605	0x003300ca00000000d900fd0000000000	0x01
10	0xdb31fb532801280602dff1f354f68a3d	0x003300ca00000000d900fd002a0000c2	0x01
11	0xdb31fb5345d645d102dff1f33921e7ea	0x003300ca00000000d900fdcc2a0000c2	0x01
12	0xdbe8c81128d80ce36d7cf1f308179d43	0x003347ca000000d5d95dfd002a000000	0x01
13	0xe47dd3610c3d280652e9ea832cf29d43	0x000047ca000000d5d900fd002a000000	0x01
14	0xff0d09870c3de990499930652cf29d43	0x003347ca000000d5d900fd0000000000	0x01
15	0xff0dc8110c3d28064999f1f32cf29d43	0x003347ca000000d5d900fd002a000000	0x01
16	0x207b6feac83b1bcdf995cb2ffa932458	0x0000000000000000d900000000000000	0x01, 0x02 or 0x20
17	0x207b7d8dc83b09aaf995cb2fe8f4363f	0x000000ca00000000d900000000000000	0x01 or 0x02
18	0x3b0b66fdc83b09aae2e5d05fe8f4363f	0x003300ca00000000d900000000000000	0x01 or 0x02
19	0x3b0b749ac83b1bcde2e5d05ffa932458	0x0033000000000000d900000000000000	0x01 or 0x02
20	0x9ef5749ac83b0fb9f69161d55f6d302c	0xb8330000000000000000000000000000	0x01, 0x02 or 0x04
21	0xea8ab50cc83bba5082eea0435f6d302c	0xb833000019000000000000002a000000	0x01 or 0x02
22	0x8585749ac83bbe335c6bd05f441d81a6	0x0062000000000000000000000000000000	0x02
23	0x97e266fdda5cac545c6bd05f567a81a6	0x0062004f0000000000000000000000000	0x02
24	0x97e286c7da5cac545c6bd05f567a619c	0x0062004f0000000000000820000000000	0x02
25	0x85858fd0c83bbe335c6bcb2f5f6d619c	0x0000000000000000000820000000000	0x01, 0x02 or 0x10
26	0xeaff0090da5cd1494e0ccb2f227081a6	0x00004b5400000000000000000000000	0x04
27	0x65bfae7cc83b9f9f7dc72b15bf57619c	0x0000000000000000000f9001a000000	0x08
28	0x880d56417b40b585a3ee75dc74f12425	0xb98800000000001a0058f9001aee00e9	0x08
29	0xd435ae7cc83b2e15cc4d9a9fbf57d016	0xb9000000000000000000f9001a000000	0x08
30	0xd435ae7cc83b3565cc4d9a9fbf57cb66	0xb9880000000000000000f9001a000000	0x08
31	0xd9bdd4df0cbe9dbb23dbe8a7aa07b049	0x0000003a00000000000000bc000046e9	0x08
32	0xd9bdd4df6169f06c4e0c8570aa07dd9e	0x0000003a000000000000000000046e9	0x08
33	0x340f6feac83b14c9ede17aa55f6d2b5c	0xb9880000000000000000000000000000	0x02 or 0x08
34	0x65bf6feac83b5e09bc512b15bf57619c	0x0000000000000000000f90000000000	0x04 or 0x08
35	0xd9bd7d8dc83bf06c4e0c85700355dd9e	0x0000003a0000000000000000000000e9	0x02 or 0x08
36	0x8585c6b8522b4b595c6bcb2f30171bb6	0x0000c4000000000000000000000008e8c	0x08 or 0x80

The faults were sorted according to the value of ΔIn. As we can see, some faults have several possibilities for the value of ΔIn. However, its correct value can be retrieve from the others faults. Indeed, the fault 16 for example at the same difference output on the 9-th byte as the fault 15. Thereby, the correct value of ΔIn for the fault 16 is 0x01. Like this, we can retrieve the correct value of ΔIn for each fault - the last fault can not have been obtained from $\Delta In =$ 0x08 since the correct value for the last nibble is this case is 0xe9. Finally, by intersection of the obtained output differences for each value of ΔIn, we obtain the following exploitable differentials on the inverse S-boxes:

(0xb83347ca19002ad5d95dfdcc2a548ac2, 0x01010101010001010101010101010101),
(0x0062004f0000000000000820000000000, 0x0002000200000000000000020000000000),
(0x00004b5400000000000000000000000, 0x00000404000000000000000000000000),
(0xb988003a0000001a0058f9bc1aee46e9, 0x0808000800000008000808080808080808),
(0x0000c4000000000000000000000008e8c, 0x0000800000000000000000000000008080).

Finally, Table 4 gives for each $1 \leq i \leq 16$, the value of the output byte difference $\Delta Y_{24}[i]$ that the 7 faults which give as much information as all faults have allowed us to know for each obtained input difference $\Delta X_{24}[i] = 2^j$ denoted ΔIn (i.e. the intersection of the obtained output differences). Table 4 also gives the byte candidates obtained for $\mathcal{L}\mathsf{layer}^{-1}(C \oplus K \oplus T)[i] \oplus \mathcal{C}t_{23}[i]$ (equal to 0x030e2eef32dbfbcbdb3f4859d1e49e97). The symbol $\emptyset$ means that the faults did not provide any information about the byte (i.e. the 256 values are possible).

Table 4: Values of $\Delta Y_{24}[i]$, $1 \leq i \leq 16$, for each obtained input difference ΔIn and bytes candidates obtained for $\mathcal{L}\mathsf{layer}^{-1}(C \oplus K \oplus T)[i] \oplus \mathcal{C}t_{23}[i]$

		ΔIn			$\mathcal{L}\mathsf{layer}^{-1}(C \oplus K \oplus T)[i] \oplus \mathcal{C}t_{23}[i]$
		0x01	0x04	0x08	
$\Delta Y_{24}[i]$	$i = 1$	0xb8	$\emptyset$	0xb9	0x03
	$i = 2$	0x33	$\emptyset$	0x88	0x0e
	$i = 3$	0x47	0x4b	$\emptyset$	0x2e
	$i = 4$	0xca	0x54	0x3a	0xef
	$i = 5$	0x19	$\emptyset$	$\emptyset$	0x2b, 0x32, 0x4f, 0x56, 0x65 or 0x7c
	$i = 6$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
	$i = 7$	0x2a	$\emptyset$	$\emptyset$	0xd1 or 0xfb
	$i = 8$	0xd5	$\emptyset$	0x1a	0xcb
	$i = 9$	0xd9	$\emptyset$	$\emptyset$	0x02 or 0xdb
	$i = 10$	0x5d	$\emptyset$	0x58	0x3f
	$i = 11$	0xfd	$\emptyset$	0xf9	0x48
	$i = 12$	0xcc	$\emptyset$	0xbc	0x59
	$i = 13$	0x2a	$\emptyset$	0x1a	0xd1
	$i = 14$	0x54	$\emptyset$	0xee	0xe4
	$i = 15$	0x8a	$\emptyset$	0x46	0x9e
	$i = 16$	0xc2	$\emptyset$	0xe9	0x97