A First DFA on PRIDE: from Theory to Practice

Benjamin Lac[1,5], Marc Beunardeau[2,6], Anne Canteaut[3], Jacques J.A. Fournier[1], and Renaud Sirdey[4]

[1] CEATech/DPACA, Gardanne, France,
[2] Ingenico Labs, Paris, France,
[3] Inria, Paris, France,
[4] CEATech/LIST, Saclay, France,
[5] ENSM-SE, Saint-Étienne, France,
[6] ENS, Paris, France,
{benjamin.lac, jacques.fournier, renaud.sirdey}@cea.fr,
marc.beunardeau@ingenico.com, anne.canteaut@inria.fr

Abstract. PRIDE is one of the most efficient lightweight block cipher proposed so far for connected objects with high performance and low-resource constraints. In this paper we describe the first ever complete Differential Fault Analysis against PRIDE. We describe how fault attacks can be used against implementations of PRIDE to recover the entire encryption key. Our attack has been validated first through simulations, and then in practice on a software implementation of PRIDE running on a device that could typically be used in IoT devices. Faults have been injected using electromagnetic pulses during the PRIDE execution and the faulty ciphertexts have been used to recover the key bits. We also discuss some countermeasures that could be used to thwart such attacks.

Keywords: lightweight cryptography · DFA · PRIDE · EM fault attacks.

1 Introduction

With the emergence of the Internet of Things (IoT), new cryptographic primitives are needed to suit the high performance, low power and low resource constraints of IoT devices. Ciphers like AES, which are good enough for 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, like for example PRESENT [10], PRINCE [12], SIMON [6] or SPECK [6]. Among those, the NSA proposal SPECK is a highly efficient software-oriented cipher, but it does not have any 'linear diffusion layer' implying that it requires a huge number of rounds to guarantee an appropriate security level. In order to keep a small number of rounds, the PRIDE cipher [4] exploits an optimal linear layer which provides a high diffusion and has highly efficient implementations. Although hardware implementations are more efficient in terms of clock cycles than software implementations, design and study of software-oriented ciphers is nevertheless important since these implementations are used in practice because they are less expensive and more flexible than hardware implementations. To date, when looking at software implementations, PRIDE is one of the most efficient

lightweight cryptographic ciphers as shown the performance comparisons given in [4,5]. This led us to study the security provided by PRIDE and its resistance to malicious attacks. In terms of security, two of the differential attacks proposed so far in the literature do not allow to recover the entire key [37,38], while a third one [14] does achieve this but under stringent conditions. Since PRIDE is to be used in IoT devices in pervasive environments, we ought to also look at implementation-related issues. In that respect, we propose in this paper the first Differential Fault Analysis (DFA) on PRIDE. 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 algorithm. It is closely related to differential cryptanalysis, but much more efficient since it exploits differential characteristics on very few rounds only.

In this paper, we first present PRIDE before describing the theoretical DFA using different fault models. Then we validate our hypotheses and equations using data onto which fault models have been 'simulated'. In order to validate the practical feasibility of our attack, we used electromagnetic pulses to inject faults during the execution of the PRIDE cipher running on an off-the-shelf chip embedding an ARM Cortex-M3 micro-controller and applied our DFA on the corrupted results obtained. So as to taking advantage of the 32-bit architecture of the micro-controller, we have implemented PRIDE in ARM assembly language. Thereby, we show the practical feasibility of our attack from 32-bit random faults. Finally we discuss countermeasures that can be implemented to thwart such attacks before concluding the paper with some perspectives.

2 Fault attacks against cryptographic algorithms

2.1 Physical attacks

Unlike mathematical attacks which target the actual definition of a cryptographic cipher, physical attacks target the way the cipher is implemented. Physical attacks can be divided into two categories: invasive and non-invasive ones. In this paper, we further focus on non-invasive techniques which mainly consist either in analysing side-channel information leakages or in injecting faults during a cryptographic computation.

Side-Channel Analyses [23], [27] exploit the fact that some physical values or "side channels" such as the power consumption [22], the electromagnetic radiation [17], [31] or the calculation time [16], [21] of an integrated circuit depend on the operations and data manipulated during a given computation. Information about the internal processes of the chip and the data it is manipulating can be derived by observing such external physical characteristics. Such analyses can be quickly mounted with cheap equipment, without altering the physical integrity of the circuit. This dependency between the side channels and the internal computations can be analysed to infer information about the data manipulated using mathematical tools like correlation [13], mutual information [18], variance [25] or entropy [26] or using architecture-dependant behaviours such as cache accesses [7], [29,30] or using branch predictions [2,1].

2.2 Fault attacks

Fault Attacks, introduced in [11], consist in disturbing the behaviour of the circuit in order to alter the correct progress of the cipher. The faults are injected into the device by various means such as light pulses [34], laser [33], clock glitches [3], spikes on the voltage supply [9] or electromagnetic (EM) perturbations [15]. Some of those techniques, like the one using a laser, 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. However it is a very costly 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 have to be made before getting the faults required by theoretical attacks. EM glitches on the other hand allow to have relatively high spatial and temporal precisions using equipment at "affordable costs" [15].

One of the objectives of fault attacks, especially when considering cryptographic ciphers, is to perform Differential Fault Analysis (DFA). DFA, originally described in [8], consists in retrieving a cryptographic key by comparing the correct ciphertexts with the faulty ones. DFA techniques have been described and applied to most publicly known cryptographic ciphers going from symmetric-key algorithms like the DES [8] or the AES [32] to asymmetric algorithms like RSA [11] or even more complex schemes like pairing-based cryptography [24]. In the particular field of lightweight cryptography, differential fault attacks have been proposed against ciphers like PRESENT [39] (used in conjunction with a cube attack), SPECK [36] (although about a hundred faults are needed which is way more than usual), TRIVIUM [28] or PRINCE [35]. The latter PRINCE block cipher has an SPN structure similar to PRIDE and in that respect the DFA proposed in [35] is quite similar to the one proposed hereafter: in our case the attack is not only adapted to the PRIDE cipher but has also been validated in practice on an embedded device running PRIDE.

DFA techniques are very efficient in retrieving the keys used during a cryptographic computation, 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 the performance of the targeted cryptographic cipher. For all those reasons, in our approach of analysing the security of implementations of PRIDE, we decided to first focus on its resistance against fault attacks in order to identify possible attack paths and devise the most efficient countermeasures in order to keep the high performance characteristics of the original cipher.

3 The PRIDE block cipher

PRIDE is an iterative block cipher composed of 20 rounds and introduced by Albrecht & al. [4] in 2014. It takes as input a 64-bit block and uses a 128-bit key $k = k_0||k_1$. The first 64 bits k_0 are used for pre- and post-whitening. The

last 64 bits k_1 are used by a key schedule to produce the subkeys $f_r(k_1)$ for each round r. The key schedule simply adds round-constants to parts of the key.

We denote k_{1_i} the i-th byte of k_1 then

$$f_r(k_1) = k_{1_0}||g_r^{(0)}(k_{1_1})||k_{1_2}||g_r^{(1)}(k_{1_3})||k_{1_4}||g_r^{(2)}(k_{1_5})||k_{1_6}||g_r^{(3)}(k_{1_7})$$

for round r with

$$\begin{aligned} g_r^{(0)}(x) &= (x + 193r) \mod 256 \\ g_r^{(1)}(x) &= (x + 165r) \mod 256 \\ g_r^{(2)}(x) &= (x + 81r) \mod 256 \\ g_r^{(3)}(x) &= (x + 197r) \mod 256 \end{aligned}$$

In this paper, $X[n]$ denotes the n-th nibble (4 bits) of a binary word X while $X\{b\}$ denotes its b-th bit. Moreover, the bits and nibbles are numbered from left to right starting from 0. The following notation is used for the intermediate values of the state within the round function $\mathcal{R}$ of PRIDE (see Figure 2):

I_r	the input of the r-th round
X_r	the state after the key addition layer of the r-th round
Y_r	the state after the substitution layer of the r-th round input
Z_r	the state after the permutation layer of the r-th round
W_r	the state after the matrix layer of the r-th round
O_r	the output of the r-th round

The r-th round, $1 \leq r \leq 19$, of PRIDE is then composed of the following steps (see Figure 2).

i. Apply the inverse permutation layer $\mathcal{P}^{-1}$ given in [4] to $f_r(k_1)$ and XOR the permuted round subkey to the input state: $X_r = I_r \oplus \mathcal{P}^{-1}(f_r(k_1))$,

ii. Apply the S-box $\mathcal{S}$ given in Table 1 to each of the 16 nibbles of X_r (i.e. apply the substitution layer $\mathcal{S}-\mathsf{layer}$ to X_r): $Y_r = \mathcal{S}-\mathsf{layer}(X_r)$,

iii. Apply the permutation layer $\mathcal{P}$ to Y_r: $Z_r = \mathcal{P}(Y_r)$,

iv. Multiply vector $\begin{pmatrix} Z_r\{16i\} \\ \vdots \\ Z_r\{16i+15\} \end{pmatrix}$ by $\mathcal{L}_i$ given in [4] for $i \in \{0, \cdots, 3\}$:

$$W_r = \mathcal{L}_0 \begin{pmatrix} Z_r\{0\} \\ \vdots \\ Z_r\{15\} \end{pmatrix} ||\mathcal{L}_1 \begin{pmatrix} Z_r\{16\} \\ \vdots \\ Z_r\{31\} \end{pmatrix} ||\mathcal{L}_2 \begin{pmatrix} Z_r\{32\} \\ \vdots \\ Z_r\{47\} \end{pmatrix} ||\mathcal{L}_3 \begin{pmatrix} Z_r\{48\} \\ \vdots \\ Z_r\{63\} \end{pmatrix},$$

v. Apply the inverse permutation $\mathcal{P}^{-1}$ to W_r: $O_r = \mathcal{P}^{-1}(W_r)$.

Table 1: S-box of the block cipher PRIDE

x	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
$\mathcal{S}(x)$	0x0	0x4	0x8	0xf	0x1	0x5	0xe	0x9	0x2	0x7	0xa	0xc	0xb	0xd	0x6	0x3

For the final round, denoted by $\mathcal{R}'$, only the first two steps are applied.

In order to encrypt a plaintext M, the cipher applies $\mathcal{P}^{-1}$ to M, then performs an XOR between the result and k_0. It then applies the 20 rounds as previously described and performs again an XOR with k_0. Finally, $\mathcal{P}$ is applied to the result to obtain the ciphertext C. Figure 1 shows the general structure of PRIDE.

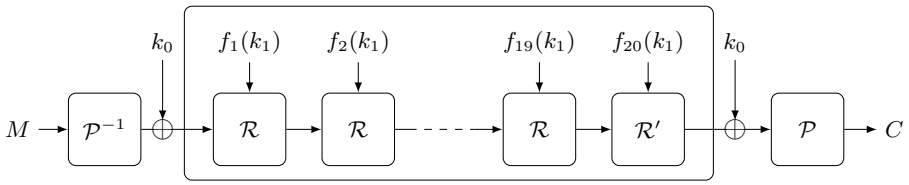

Figure 1: The structure of PRIDE

The PRIDE round function $\mathcal{R}$ is depicted on Figure 2.

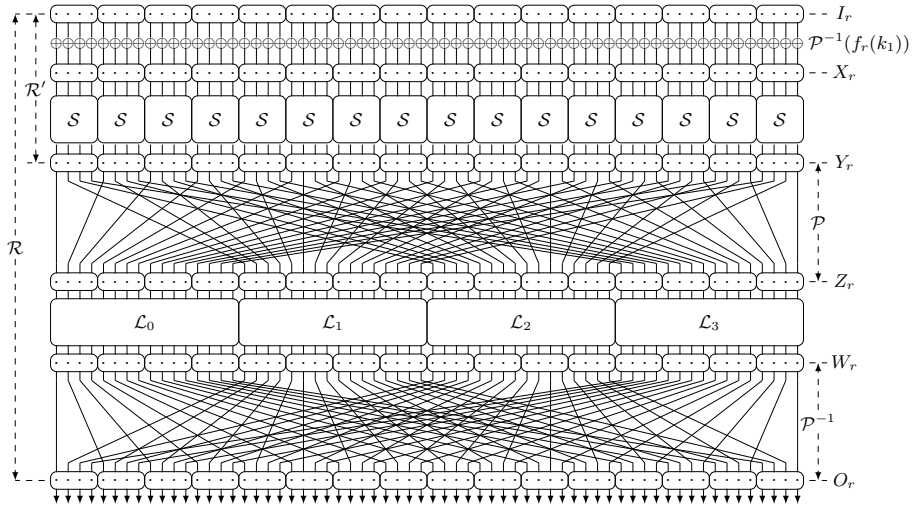

Figure 2: The PRIDE round function

4 Differential Fault Analysis of PRIDE

In this section, we present a technique adapted from the proposed attack in [35] to retrieve the secret key using fault injections on PRIDE computations. The aim of our analysis is to minimize the number of fault injections needed. We use ideal fault models and we describe how to exploit them to retrieve the key.

4.1 General principle

Despite their similarities, a DFA 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 they can be injected at the moments where the attacker wants.

The DFA that we propose in this paper also differs from most classical DFA since it is not based on statistical methods: it is deterministic.

The attack is composed of two stages, one consists in corrupting data manipulated in the penultimate round to retrieve k_0 and the other in attacking the antepenultimate round to retrieve k_1. The general structure of the attack is to exploit the diffusion of a 16-bit word within the inverse permutation layer in order to get a known 4-bit difference at the input of each S-box on the following round. Together with the knowledge of the output difference of each S-box, which are derived from the correct and faulty ciphertexts, C and C^*, this allows us to retrieve information about the key. To this end, we exploit the difference distribution table of the PRIDE S-box given in Appendix A. Indeed, obtaining information on k_0 is possible from the following equation:

$$\Delta X_{20} = \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C^*) \oplus k_0),$$

where $\mathcal{S}\text{−}\mathsf{layer} = \mathcal{S}\text{−}\mathsf{layer}^{-1}$ denotes the substitution layer. We can use this equation for each nibble $0 \leq i \leq 15$:

$$x = \mathcal{P}^{-1}(C)[i] \oplus k_0[i] \text{ and } y = \mathcal{P}^{-1}(C^*)[i] \oplus k_0[i] \text{ satisfy}$$

$$x \oplus y = \Delta Y_{20}[i] = \mathcal{P}^{-1}(\Delta C)[i] \text{ and } \mathcal{S}^{-1}(x) \oplus \mathcal{S}^{-1}(y) = \Delta X_{20}[i].$$

From the knowledge of a nonzero input difference $\Delta Y_{20}[i]$ and of an output difference $\Delta X_{20}[i]$ for $\mathcal{S}^{-1}$, we deduce 2 or 4 candidates for the input value x, because the differential uniformity of $\mathcal{S}^{-1}$ equals 4 (see the difference distribution table in Appendix A). Moreover, Proposition 1 enables us to exhibit 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 with differential uniformity 4. 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 $(a_1 \oplus a_2, b_1 \oplus b_2)$ equals to 2, then we can guarantee that the input satisfying these two differentials simultaneously is unique.

Note: if one of the three equations does not have any solution, then the system of two equations (1) and (2) does not have any solution neither.

Once k_0 has been recovered (we will see in the next parts some strategies to achieve this end), X_{20} and X^*_{20} can be computed from the ciphertexts C and C^*. Let $\mathcal{L}$ denote the whole linear layer, i.e.,

$$\mathcal{L} = \mathcal{P}^{-1} \circ \begin{pmatrix} \mathcal{L}_0 & 0 & 0 & 0 \\ 0 & \mathcal{L}_1 & 0 & 0 \\ 0 & 0 & \mathcal{L}_2 & 0 \\ 0 & 0 & 0 & \mathcal{L}_3 \end{pmatrix} \circ \mathcal{P}.$$

Then ΔY_{19} can be computed and the following equation

$$\begin{aligned} \Delta X_{19} = \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))) \\ \oplus \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C^*) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))), \end{aligned}$$

allows the attacker to recover $\mathcal{P}^{-1}(f_{20}(k_1))$ and therefore k_1, with the same method but from fault injections in the 18-th round. Indeed, for $0 \leq i \leq 15$:

$$\begin{gathered} x = \mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))[i] \text{ and} \\ y = \mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C^*) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))[i] \text{ satisfy} \end{gathered}$$

$$\begin{gathered} x \oplus y = \Delta Y_{19}[i] = \mathcal{L}^{-1}(\mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C \oplus k_0)) \oplus \mathcal{S}\text{−}\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C^* \oplus k_0)))[i] \\ \text{and } \mathcal{S}^{-1}(x) \oplus \mathcal{S}^{-1}(y) = \Delta X_{19}[i]. \end{gathered}$$

4.2 Ideal fault model

The strategies we propose require at least 2 fault injections for each stage of the attack to retrieve a round key (i.e 4 to retrieve the complete key). For the first stage, whose objective is to find k_0, one of the following approaches can be used:

$$\text{(i.) Flip } Z^0_{19} \text{ then } Z^3_{19} \text{ or (ii.) Flip } W^0_{19} \text{ then } W^3_{19},$$

where Z^i_r (resp. W^i_r) denotes the input (resp. output) of the matrix $\mathcal{L}_i$ at round r. Then, to retrieve the key k_1, and so the complete key, the possible fault injections are the same but are carried out on Z_{18} or W_{18}. A flip of Z^0_r gives us a difference equal to 0xffff on the input of the matrix $\mathcal{L}_0$. The matrix being linear, we know that the output difference is also 0xffff. The latter being the same value than the one obtained with a flip of W^0_r. The other matrices have differences in input and output equal to zero. Then, the inverse permutation layer also being linear, we know the input difference of each S-box of the substitution layer at round $r+1$. These values are equal to 0x8, so we obtain $\Delta X_{r+1}[i] = 0\text{x}8$ for all $i \in \{0, \cdots, 15\}$. Moreover, we recall that the output differences are known from the correct and faulty ciphertexts. Figure 3 shows the propagation of the difference (displayed in red) obtained by a flip of Z^0_{19}. In the same way, a flip of Z^3_r or W^3_r yields a difference of 0x1 on each S-box at round $r+1$. Finally, with strategy (i.) or (ii.), we obtain pairs of differentials $(\Delta Y_{20}[i], \Delta X_{20}[i])_1 = (a_1, 0\text{x}1)$ and $(\Delta Y_{20}[i], \Delta X_{20}[i])_2 = (a_2, 0\text{x}8)$ for all $i \in \{0, \cdots, 15\}$ with a_1 and a_2 known. We get the same pairs for $(\Delta Y_{19}[i], \Delta X_{19}[i])$ from faults on the 18-th round. Since $0\text{x}1 \oplus 0\text{x}8 = 0\text{x}9$, from the Proposition 1 (and the difference distribution table in Appendix A),

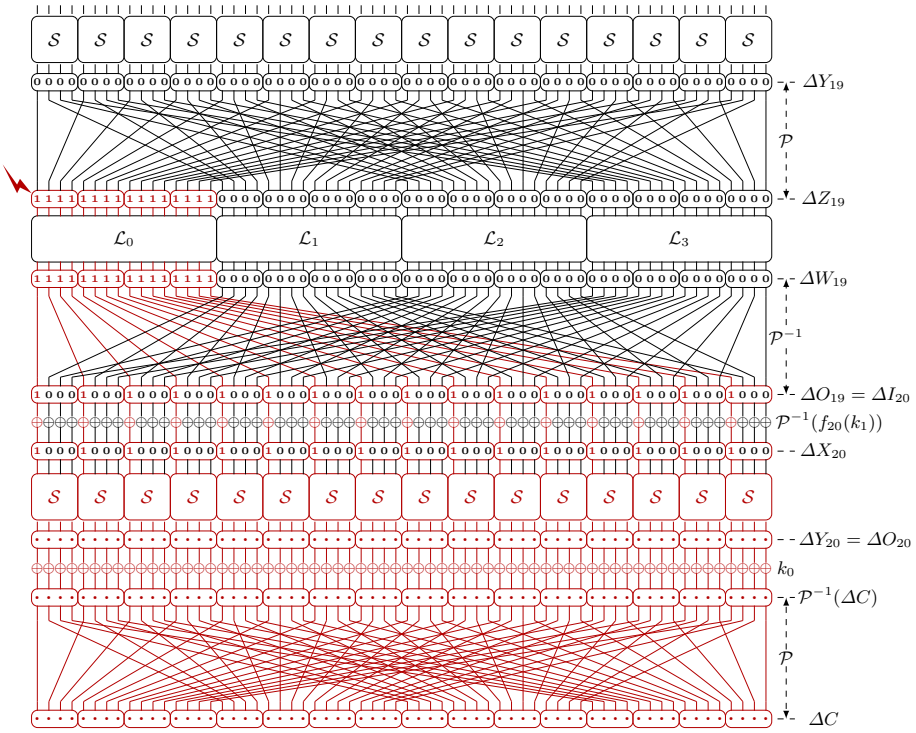

Figure 3: Propagation on PRIDE of the difference obtained by a flip of Z_{19}^0

there is only one element in the intersection of the two sets of solutions obtained for each nibble. Therefore, we have shown that we get only one candidate for each nibble of $x = \mathcal{P}^{-1}(C) \oplus k_0$ from faults on the 19-th round and one candidate for each nibble of $x = \mathcal{L}^{-1}(\mathcal{S}-\mathsf{layer}^{-1}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))[i]$. Finally, from the knowledge of C we retrieve k_0 and from the key schedule we retrieve k_1.

The strategies we have presented require 4 fault injections to retrieve the complete key. In case the attacker obtains fewer faults, Table 2 shows the time complexity, expressed as a number of encryptions, that an attacker can obtain to retrieve the secret key k with 1 to 3 faults following the ideal fault model. A proof of these values is given in Appendix B.

Table 2: Trade-offs between the time complexity, expressed as a number of encryptions, and the number of faults with the ideal fault model.

Number of faults	1	2	3
Time complexity	2^{64}	2^{32}	$2^{27.7}$

4.3 Random fault model

In order to achieve the attack, we must flip all the bits of four 16-bit words for the ideal fault model used in the preceding part. However, we can see that

reversing one bit provides an active S-box, it is therefore enough to inverse all the bits of the desired 16-bit words. Indeed, if we flip the bit i of W_{19}^0 from one fault, we obtain 4 candidates for the nibble i of the subkey k_0. Moreover, if we flip the bit i of W_{19}^3 from an other fault, we retrieve (by intersection) the value of the nibble i of k_0.

It is easy to target a specific instruction from a simple power (or EM) analysis for example in practice. If the instruction is less than 16 bits, we can then reduce the key space from each active S-box, until it is enough small for an exhaustive search. Finally, we will see in the section 5 that the attack is still effective from 32-bit faults, only the exploitation of the faults is different.

4.4 Properties exploited by our attack

Our attack mainly exploits two properties of the building-blocks of PRIDE:

The design of the linear layer based on the so-called interleaved construction. Indeed, this construction aims at designing a diffusion layer with a high branch number (see Theorem 1 in [4]). For a SPN whose substitution layer is composed of n S-boxes over $\mathbb{F}_2^k$, the linear layer obtained by the interleaved construction is defined as $L = \mathcal{P}^{-1} \circ \mathcal{L} \circ \mathcal{P}$ where $\mathcal{P}$ is an isomorphism from $(\mathbb{F}_2^k)^n$ into $(\mathbb{F}_2^n)^k$. Then, we deduce from the definition of $\mathcal{P}$ that flipping the n bits of any word at the input of $\mathcal{P}^{-1}$ in $W = (W_1, \cdots, W_k)$ activates all S-boxes in the next round. Indeed, by construction, the n bits of any W_i go to different S-boxes. Hence flipping n consecutive bits in the linear layer of the penultimate round allows the attacker to recover information on all the n nibbles of the subkey used in the last round. The number of candidates for this last-round subkey is upper-bounded by $\delta(\mathcal{S})^n$, where $\delta(\mathcal{S})$ is the differential-uniformity of the S-box ($\delta(\mathcal{S}) = 4$ in the case of PRIDE and of most block ciphers using 4-bit S-boxes).

The differential properties of the S-box, which avoids the existence of differentials with high probability over a large number of rounds. The counterpart of this resistance against classical differential cryptanalysis is that the number of inputs which satisfy two valid differentials simultaneously is usually reduced to a single element. This property enables the attacker to drastically reduce the number of subkey candidates. In the case of PRIDE, two faults, each on n consecutive bits in the linear layer, are enough to obtain a single candidate for the subkey.

4.5 Simulation of the DFA on PRIDE

In order to validate our theoretical DFA against PRIDE and test the correctness of the proposed equations, we first performed a validation by simulation.

In this section we assume that a device executes PRIDE with a key $k = k_0||k_1$ where k_0 = 0xefcdab8967452301 and k_1 = 0x0123456789abcdef. We further assume that an attacker successfully flips all the bits of Z_{19}^0, Z_{19}^3, W_{18}^0 and W_{18}^3.

Then, she obtains the following ciphertexts from 5 executions of the same plaintext 0xfedcba9876543210:

i. 0xc40f2551f39c63a9 the correct ciphertext,
ii. 0xe7f325510dc3b7a8, 0xc40fdaaec89376f7 from a flip of Z_{19}^0, Z_{19}^3,
iii. 0x2857589433cbdead, 0x461720d9729c1956 from a flip of W_{18}^0, W_{18}^3.

The knowledge of the plaintext is not necessary, it is sufficient to ensure that the same plaintext is used for each execution.[7] The attacker obtains the following differentials for the last substitution layer from the first two faulty ciphertexts:

i. $(\Delta X_{20}, \Delta Y_{20})_1$ = (0x8888888888888888, 0x33a323a88a8aaa23),
ii. $(\Delta X_{20}, \Delta Y_{20})_2$ = (0x1111111111111111, 0x4467656745457776).

From the first differential, she obtains a set of candidates for each nibble of $\mathcal{P}^{-1}(C) \oplus k_0$ where C is the correct ciphertext. She can then find a set of candidates for each nibble of k_0 from $\mathcal{P}^{-1}(C)$ = 0xab720c373416ba8d. Table 3 shows the obtained sets of candidates.

Table 3: Sets of candidates obtained from $(\Delta X_{20}, \Delta Y_{20})_1$

$k_0[0]$	$k_0[1]$	$k_0[2]$	$k_0[3]$	$k_0[4]$	$k_0[5]$	$k_0[6]$	$k_0[7]$	$k_0[8]$	$k_0[9]$	$k_0[10]$	$k_0[11]$	$k_0[12]$	$k_0[13]$	$k_0[14]$	$k_0[15]$
0x5	0x4	0x4	0x5	0x0	0x0	0x0	0x1	0x5	0x5	0x4	0x5	0x0	0x1	0x0	0x1
0x6	0x7	0x6	0x6	0x2	0x3	0x2	0x2	0x6	0x7	0x7	0x7	0x2	0x3	0x2	0x2
0xd	0xc	0xc	0xd	0x8	0x8	0x8	0x9	0xd	0xd	0xc	0xd	0x8	0x9	0x8	0x9
0xe	0xf	0xe	0xe	0xa	0xb	0xa	0xa	0xe	0xf	0xf	0xf	0xa	0xb	0xa	0xa

From the last differential, the attacker obtains another set of candidates for each nibble of k_0. Table 4 shows the resulting candidates.

Table 4: Sets of candidates obtained from $(\Delta X_{20}, \Delta Y_{20})_2$

$k_0[0]$	$k_0[1]$	$k_0[2]$	$k_0[3]$	$k_0[4]$	$k_0[5]$	$k_0[6]$	$k_0[7]$	$k_0[8]$	$k_0[9]$	$k_0[10]$	$k_0[11]$	$k_0[12]$	$k_0[13]$	$k_0[14]$	$k_0[15]$
0xa	0xa	0xa	0xa	0xa	0xa	0x8	0x8	0x2	0x2	0x0	0x0	0x2	0x2	0x0	0x0
0xb	0xb	0xb	0xb	0xb	0xb	0x9	0x9	0x3	0x3	0x1	0x1	0x3	0x3	0x1	0x1
0xe	0xe	0xc	0xc	0xc	0xe	0xe	0xe	0x6	0x6	0x4	0x4	0x4	0x4	0x6	0x6
0xf	0xf	0xd	0xd	0xd	0xf	0xf	0xf	0x7	0x7	0x5	0x5	0x5	0x5	0x7	0x7

By doing the intersection of the obtained two sets for each nibble, the attacker gets k_0. Then, with this value of k_0, she obtains the following differences for the antepenultimate substitution layer from the flip of W_{18}^0 and W_{18}^3:

i. $(\Delta X_{19}, \Delta Y_{19})_1$ = (0x8888888888888888, 0x23a2288338832828),
ii. $(\Delta X_{19}, \Delta Y_{19})_2$ = (0x1111111111111111, 0x7777456474776476).

From the first differential, she obtains sets of candidates for each nibble Nib_i of $\mathcal{L}^{-1}(\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))$ with $i \in \{0, \cdots, 15\}$. Table 5 shows the sets of candidates she gets.

[7] If it is not the case, the attacker can mount an attack if she knows, for each faulty ciphertext, the corresponding correct ciphertext - to obtain differentials for the S-boxes. But the key may not be recovered in this case since the information obtained by the attacker depends on the value of the correct ciphertext.

Table 5: Sets of candidates obtained from $(\Delta X_{19}, \Delta Y_{19})_1$

Nib_0	Nib_1	Nib_2	Nib_3	Nib_4	Nib_5	Nib_6	Nib_7	Nib_8	Nib_9	Nib_{10}	Nib_{11}	Nib_{12}	Nib_{13}	Nib_{14}	Nib_{15}
0x0	0x4	0x1	0x0	0x0	0x5	0x5	0x4	0x4	0x5	0x5	0x4	0x0	0x5	0x0	0x5
0x2	0x7	0x3	0x2	0x2	0x6	0x6	0x7	0x7	0x6	0x6	0x7	0x2	0x6	0x2	0x6
0x8	0xc	0x9	0x8	0x8	0xd	0xd	0xc	0xc	0xd	0xd	0xc	0x8	0xd	0x8	0xd
0xa	0xf	0xb	0xa	0xa	0xe	0xe	0xf	0xf	0xe	0xe	0xf	0xa	0xe	0xa	0xe

From the last differential, the attacker obtains other sets of candidates for each nibble Nib_i of $\mathcal{L}^{-1}(\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))$ with $i \in \{0, \cdots, 15\}$. Table 6 shows the sets of candidates obtained.

Table 6: Sets of candidates obtained from $(\Delta X_{19}, \Delta Y_{19})_2$

Nib_0	Nib_1	Nib_2	Nib_3	Nib_4	Nib_5	Nib_6	Nib_7	Nib_8	Nib_9	Nib_{10}	Nib_{11}	Nib_{12}	Nib_{13}	Nib_{14}	Nib_{15}
0x8	0x8	0x8	0x8	0x0	0x2	0xa	0x0	0x8	0x0	0x8	0x8	0xa	0x0	0x8	0xa
0x9	0x9	0x9	0x9	0x1	0x3	0xb	0x1	0x9	0x1	0x9	0x9	0xb	0x1	0x9	0xb
0xe	0xe	0xe	0xe	0x4	0x6	0xc	0x4	0xe	0x4	0xe	0xe	0xc	0x4	0xe	0xc
0xf	0xf	0xf	0xf	0x5	0x7	0xd	0x5	0xf	0x5	0xf	0xf	0xd	0x5	0xf	0xd

By intersecting the obtained two sets for each nibble, the attacker gets

$$\mathcal{L}^{-1}(\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1))) = \text{0x8f9806d4f5efa58d}.$$

Then, she computes

$$\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)) = \text{0x24c39cc978f41dd4}$$

and from $\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) = \text{0x11c3a9c65f5f772b}$, she retrieves

$$\mathcal{P}^{-1}(f_{20}(k_1)) = \text{0x3500350f27ab6aff}.$$

Finally she deduces $f_{20}(k_1) = \text{0x0137454b89ffcd53}$, she gets k_1 from the key scheduling and so she retrieves the complete key.

5 Practical implementation of the DFA on PRIDE

In order to test the feasibility of our attack against the PRIDE block cipher, we have implemented and run the cipher on an STM32 chip embedding an ARM Cortex-M3 micro-controller. That particular chip was chosen because it is quite representative of the off-the-shelf devices used for IoT applications. Note that the chip does not embed any countermeasures against the kind of the fault attacks implemented in this paper. We validated the attack on an implementation in ARM assembly language taking advantage of the 32-bit architecture of the micro-controller. We present in this section the full analysis conducted on this implementation. The source code is given in Appendix C and Table 7 compares the performances of this implementation with that of the implementation in AVR assembly language whose source code and performances are given in [4].

So as to inject exploitable faults into such a chip, we used EM pulses because with this approach we did not need to decapsulate the chip and we were able to

Table 7: Comparison between AVR and ARM assembly implementation

	Time (cycle)	Size (bytes)
AVR assembly implementation (given in [4])	1514	266
ARM assembly implementation (Appendix C)	2375	490

inject faults at precise enough instants to target specific instructions of the cipher during its execution. The set-up we used is quite similar to the one described in [15], with the difference that we did not need any motorized X-Y stage: injecting faults 'in the center' of the chip was good enough for having a fault model close to a random fault model (one chance over two to flip a bit). Indeed, it is possible to target a precise 32-bit word (more precisely a specific instruction) but the injected faults follow a random pattern. In order to obtain pairs of differentials $(\Delta X_{20}[i], \Delta Y_{20}[i])$ (resp. $(\Delta X_{19}[i], \Delta Y_{19}[i])$) for $i \in \{0, \cdots, 15\}$, we injected the faults on the first and on the second 32-bit word of the state before the inverse permutation in the 19-th (resp. 18-th) round. by as many faults as necessary. Each fault on the first word provided us differences on each nibble of ΔX_{20} equal to 0x0, 0x4, 0x8 or 0xc and equal to 0x0, 0x1, 0x2 or 0x3 from each fault on the second word. We validated the attack from these 32-bit faults, we will see that the faults exploitation is different (some pairs of differentials do not allow us a single candidate) but the attack is nevertheless still effective.

In our experiment, we used a key $k = k_0||k_1$ where k_0 = 0xf3f721cb1c882658 and k_1 = 0xe417d148e239ca5d. The plaintext used for all executions was 0x0132546 798badcfe and the correct ciphertext was 0x9aecb37ea45a6c89. We used a simple EM analysis to identify in time the 18-th and 19-th rounds. Figure 4 shows the curve obtained on the oscilloscope, the 20 rounds are displayed in red.

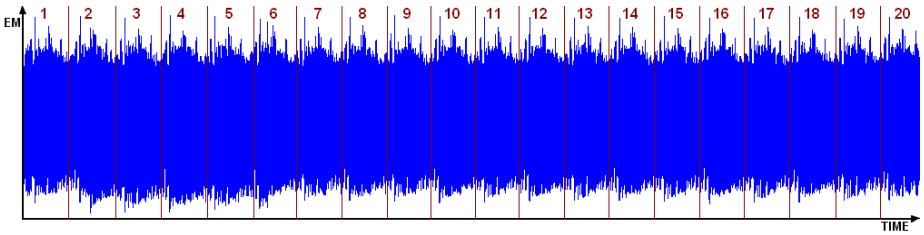

Figure 4: EM curve measured of PRIDE cipher

Then we used an electromagnetic pulse generator to disrupt the PRIDE's execution. Table 8 (resp. Table 9) shows the faults we have obtained from the electromagnetic injection on W_{19} (resp. W_{18}) numbered from 1 to 25. For each fault, Table 8 (resp. Table 9) provides the value of ΔX_{20} and ΔY_{20} (resp. ΔX_{19} and ΔY_{19}), only obtained from the correct and the faulty ciphertexts. We denote respectively by θ, β, γ, δ the possible pair of values (0x2,0x3), (0x4,0x8), (0x4,0xc), (0x8,0xc). Indeed, some differences in output of the S-boxes can be obtained from two distinct differences in input. Finally, we give in each table the fault value computed after retrieving the key.

Table 8: Faults obtained on the 19-th round

No.	Faulty ciphertext	Value of the fault on W_{19}	Value of ΔY_{20}	Value of ΔX_{20}
1	0x1aad3b972c92ec09	0x00000000804108e8	0xf00060007e40600c	0xθ0001000$1\theta$10100θ
2	0x7b4c93dea55a6d89	0x00000000e1a0a0a0	0x88c0000bc0c00000	0x$\theta\theta\theta$0000$\theta\theta$0θ000000
3	0x1b6c733e255aadc9	0x0000000081804040	0xf500000b85000000	0xθ100000$\theta\theta$1000000
4	0x71ecd27ee55a6d89	0x00000000eb00e900	0x8ec0808f00000000	0x$\theta\theta\theta$0θ0$\theta\theta$00000000
5	0x9aecb324a4426cdb	0x000000000000005a	0x0000000005076050	0x0000000001011010
6	0x9a57b33fa4626cf1	0x0000000000bb005a	0x0000000085bbb08c	0x00000000θ1$\theta\theta\theta$0$\theta\theta$
7	0x9a57b365a4606cb9	0x0000000000bb0000	0x0000000080bfe0ec	0x00000000θ0$\theta\theta\theta$0$\theta\theta$
8	0x77aa24313111ed8c	0x00000000ed461f4d	0xf8868e4f0e006de7	0x$\theta\theta\theta$1$\theta\theta$1θ0θ001$\theta\theta$1
9	0x9ae8b37ac15a6989	0x6500040400000000	0x0220030300000c00	0x0$\delta\delta$00δ0δ00000γ00
10	0x8aecb27e415abc89	0xe400d10000000000	0x3329020600000000	0x$\delta\delta\delta\gamma$0δ04000000000
11	0xa3e692ed909ee688	0x355fab9300000000	0x10ea921c620482c5	0x40c$\beta\gamma\delta$4γ4δ0c8$\delta\gamma$c
12	0x05ecb27e565a7289	0xf3001f0000000000	0xa22b99bc00000000	0x$\beta\delta\delta$c$\gamma\gamma$cγ00000000

Note: Out of 2,000 shots, we don't get any cipher for 1,219 cases and we get 247 faulty ciphers including 13 exploitable (i.e. which satisfied the conditions for our DFA). Non exploitable faulty ciphers came from a dysfunction of the UART due to the faults.

Table 9: Faults obtained on the 18-th round

No.	Faulty ciphertext	Value of the fault on W_{18}	Value of ΔY_{19}	Value of ΔX_{19}
13	0xf24690de8df8cc89	0x0000000082000000	0xc00000b000000000	0xθ0000θ0000000000
14	0x2df93aebf5935009	0x0000000041c0d0d0	0x7807000bd8050000	0x1θ01000$\theta\theta\theta$010000
15	0xa9a4a34f84604dde	0x0000000003010707	0x000004cd0000065c	0x000001$\theta\theta$00000011θ
16	0x52c367c49a9b8786	0x0000000000b55858	0x05077000b6d84808	0x01011000θ1$\theta\theta$1$\theta\theta$
17	0x00632c247f18e99e	0x0000000058580000	0x0e0bb0000d0ef000	0x0θ0$\theta\theta$0000θ0$\theta\theta$000
18	0xecbc98d50864ad3a	0x00000000a7a70000	0xc0f008bbb0d00888	0xθ0θ00$\theta\theta\theta\theta$0θ00$\theta\theta$
19	0x43b733ec34c1ec11	0x0093000000000000	0x00000000300a0022	0x00000000δ00β00$\delta\delta$
20	0xcabdf870ee423736	0x75e5575700000000	0x0c8c0b123baf049e	0x0γ8γ0c4$\delta\delta$cβ40cγc
21	0x46eb59132610ef55	0x01e0c60100000000	0x6f0001133aa00006	0x4400044$\delta\delta\beta\beta$00004
22	0x9d13b57cf2211618	0x13974cd400000000	0x0f036133290c0422	0x040δ44$\delta\delta\delta\gamma$0γ0c$\delta\delta$
23	0x1247352b2400c0ed	0x0000006700000000	0x0000000009900c96	0x000000000$\gamma\gamma$00$\gamma\gamma$4
24	0x770a084c5528c599	0x6363000000000000	0x0a8000330aa00022	0x0β8000$\delta\delta$0$\beta\beta$000$\delta\delta$
25	0xc80ca16eb67b9711	0x3600a90000000000	0x6043623a00000000	0x40cδ4$\delta\delta\beta$00000000

We now give, among the obtained faults, those that give as much information as all faults and all sets of candidates that we can extract from each fault. Table 10 shows all sets of candidates obtained for each nibble of k_0 from the differentials (ΔY_{20}, ΔX_{20}) and from $\mathcal{P}^{-1}(C)$ = 0xe17c93c49ec6fc61 with C the correct ciphertext. Symbol $\emptyset$ means that the fault does not provide any information about the nibble (i.e. the 16 values are possible).

We eventually get 4 possible values for k_0 with $k_0[8] \in \{0x0, 0x1\}$ and $k_0[10] \in \{0x8, 0x9\}$. In order to reduce the number of possible keys, we then used faulty ciphers obtained from fault injection on the 18-th round. For this, we compute the difference output ΔY_{19} from the remaining 4 candidates for the key. Then we can observe that some differentials (ΔX_{19}, ΔY_{19}) are not possible and therefore remove the corresponding candidate.

Table 10: Sets of candidates obtained from (ΔY_{20}, ΔX_{20})

No.	$k_0[0]$	$k_0[1]$	$k_0[2]$	$k_0[3]$	$k_0[4]$	$k_0[5]$	$k_0[6]$	$k_0[7]$	$k_0[8]$	$k_0[9]$	$k_0[10]$	$k_0[11]$	$k_0[12]$	$k_0[13]$	$k_0[14]$	$k_0[15]$
1	0x0 0x1 0xe 0xf	∅	∅	∅	0x2 0x3 0x4 0x5	∅	∅	∅	0x0 0x1 0x6 0x7	0x2 0x3 0xc 0xd	0x8 0x9 0xc 0xd	∅	0x2 0x3 0x4 0x5	∅	∅	0x4 0x5 0x8 0x9
3	0x0 0x1 0xe 0xf	0x2 0x3 0x6 0x7	∅	∅	∅	∅	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x8 0x9 0xc 0xd	∅	∅	∅	∅	∅	∅
6	∅	∅	∅	∅	∅	∅	∅	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x8 0x9 0xc 0xd	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	∅	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x4 0x5 0x8 0x9
8	0x0 0x1 0xe 0xf	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x0 0x1 0x6 0x7	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0xe 0xf	0x8 0x9 0xc 0xd	0x4 0x5 0xa 0xb	∅	0x2 0x3 0xc 0xd	∅	∅	0x2 0x3 0x4 0x5	0x6 0x7 0xa 0xb	0x4 0x5 0xa 0xb	0x8 0x9 0xe 0xf
11	0xa 0xb 0xe 0xf	∅	0x1 0xf	0x1 0x5 0x7 0xb 0xd 0xf	0x2 0x4 0xb 0xd	0x1 0x3 0x4 0x6 0x9 0xb	0x8 0x9 0xc 0xd	0x2 0x7 0xb 0xe	0x0 0x1 0x6 0x7	0x4 0x6 0x9 0xb 0xc 0xe	∅	0x8 0xc	0x1 0x2 0x9 0xa	0x4 0x6 0x9 0xb 0xc 0xe	0x0 0x5 0x9 0xc	0x8 0xd
12	0x3 0x5 0x7 0x9 0xd 0xf	0x1 0x3 0x4 0x6 0x9 0xb	0x0 0x2 0x5 0x7 0xd 0xf	0x7 0xc	0x2 0x4 0xb 0xd	0x1 0x7 0x8 0xe	0x7 0xc	0x2 0x7 0xb 0xe	∅	∅	∅	∅	∅	∅	∅	∅

Indeed, from the faulty ciphertext 0xf24690de8df8cc89 obtained from a fault on W_{18}, we obtain the 4 following values for ΔY_{19} for each possible value of k_0:

k_0	ΔY_{19}
f3f721cb0c882658	0xc000009022000000
f3f721cb0c982658	0xe000009022220000
f3f721cb1c882658	0xc00000b000000000
f3f721cb1c982658	0xe00000b000220000

and since we know that we injected faults on the last 32 bits of W_{18}, we know that each nibble of ΔX_{19} is either 0x0, 0x1, 0x2 or 0x3. From the difference distribution table of the S-box, we see that an input difference equal to 0x1, 0x2 or 0x3 can lead to an output difference in {0x4, 0x5, 0x6, 0x7, 0x8, 0xb, 0xc, 0xd, 0xe, 0xf} only. Consequently, we retrieve k_0 (displayed in red).

Then, Table 11 shows all sets of candidates obtained for each nibble Nib_i of $\mathcal{L}^{-1}(\mathcal{S}(\mathcal{P}^{-1}(C) \oplus k_0) \oplus \mathcal{P}^{-1}(f_{20}(k_1)))$ with $i \in \{0, \cdots, 15\}$, from differentials (ΔY_{19}, ΔX_{19}). We again denote by $\emptyset$ when the fault does not provide any information about the nibble (i.e. the 16 values are possible).

Finally, by intersecting sets for each nibble, we deduce 8 candidates for k_1 from k_0 and C and we retrieve the correct value of k by testing all. With this we provide, to the best of our knowledge, the first practical validation of a DFA against PRIDE, even against any light weight SPN-block cipher.

Note : We observed that injecting 32-bit random faults allows us to have lower complexity than with 16-bit random faults. Indeed, although the differential pairs obtained do not always provide a single candidate in the case of 32-bit faults, the

Table 11: Sets of candidates obtained from (ΔY_{19}, ΔX_{19})

No.	Nib_0	Nib_1	Nib_2	Nib_3	Nib_4	Nib_5	Nib_6	Nib_7	Nib_8	Nib_9	Nib_{10}	Nib_{11}	Nib_{12}	Nib_{13}	Nib_{14}	Nib_{15}
16	∅	0x2 0x3 0x6 0x7	∅	0x8 0x9 0xe 0xf	0x8 0x9 0xe 0xf	∅	∅	∅	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0xa 0xb 0xc 0xd	0x6 0x7 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x4 0x5	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb
17	∅	0x2 0x3 0xa 0xb	∅	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	∅	∅	∅	∅	0x6 0x7 0xa 0xb	∅	0x2 0x3 0xa 0xb	0x0 0x1 0xe 0xf	∅	∅	∅
18	0x4 0x5 0x8 0x9	∅	0x0 0x1 0xe 0xf	∅	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	0x4 0x5 0x6 0x7 0xc 0xd 0xe 0xf	∅	0x6 0x7 0xa 0xb	∅	∅	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb	0x0 0x1 0x2 0x3 0x8 0x9 0xa 0xb
20	∅	0x3 0x6 0xa 0xf	0x5 0x6 0xd 0xe	0x3 0x6 0xa 0xf	∅	0x0 0xb	0x0 0x1 0x4 0x5	0x0 0x2 0x5 0x7 0x8 0xa	0x1 0x2 0x4 0x7 0xc 0xf	0x0 0xb	0x1 0x3 0x7 0x9 0xb 0xd	0x3 0xc	∅	0xa 0xe	0x2 0x4 0xb 0xd	0x6 0x8
22	∅	0x3 0xc	∅	0x1 0x2 0x4 0x7 0xc 0xf	0x8 0x9 0xe 0xf	0x0 0x1 0x4 0x5	0x1 0x2 0x4 0x7 0xc 0xf	0x1 0x2 0x4 0x7 0xc 0xf	0x0 0x2 0x5 0x7 0x8 0xa	0x2 0x4 0xb 0xd	∅	0x3 0x6 0xa 0xf	∅	0xa 0xe	0x0 0x2 0x5 0x7 0x8 0xa	0x0 0x2 0x5 0x7 0x8 0xa
23	∅	∅	∅	∅	∅	∅	∅	∅	∅	0x2 0x4 0xb 0xd	0x2 0x4 0xb 0xd	∅	∅	0x3 0x6 0xa 0xf	0x2 0x4 0xb 0xd	0x8 0x9 0xe 0xf
25	0x8 0x9 0xe 0xf	∅	0xa 0xe	0x1 0x2 0x4 0x7 0xc 0xf	0x8 0x9 0xe 0xf	0x0 0x2 0x5 0x7 0x8 0xa	0x1 0x2 0x4 0x7 0xc 0xf	0x1 0x3 0x7 0x9 0xb 0xd	∅	∅	∅	∅	∅	∅	∅	∅

probability to obtain a differential is greater than with 16-bit faults. Finally we showed that flipping one bit give us a known difference on a nibble, we can so lead the attack with faults from 1 to 32 bits.

6 Countermeasures

In this section, we present and briefly analyze three possible countermeasures. This list of countermeasures is not exhaustive and any combination of those three can be used in practice to thwart the DFA proposed in this paper.

6.1 Duplication of computations

Description: A simple countermeasure is to make two computations for the last two rounds. We save the state of the cipher W_{17} in memory, possibly k times for more security - since we are in lightweight cryptography is seems reasonable to take $k = 1$ or $k = 2$. Then we make the computations up to O_{20} and save the state again. We repeat the computation with the saved state (W_{17}) and compare with the first result - possibly k times again. If two different computations give different results we trap the cipher and no output is produced by the system. Else the execution performs normally.

Cost: This countermeasure uses, for encryption and decryption, two additional matrix layer and three additional substitution layers, subkey updates and subkey additions. The cost can be bounded from above by 15% of the total PRIDE cost.

6.2 Desynchronization

Description: This countermeasure consists in adding time randomization during the cipher so that the temporal position of the 18-th and the 19-th round will not be the same for each execution. For the time randomization generation we can use a simple Linear Feedback Shift Register (LFSR) whose value indicates the 'random' delay time. Those random delay functions can be added before the 18-th round.

Cost: The cost depends on the time randomization generation - a simple LFSR implemented in hardware has a low cost with respect to IoT constraints, it also depends on the duration of the 'random delay' and on the time needed to access the random output of the LFSR.

6.3 Masking

Description: Another countermeasure proposed by Guilley *and al.* in [20] 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 64-bit random mask different at each execution, XOR it with the asked plaintext and the ciphertext obtained is sent with the mask.

In our case, we use a simple LFSR defined by a minimal primitive polynomial of degree 64 ($X^{64} + X^{63} + X^{61} + X^{60} + 1$ for example) and by an initialization made public. The LFSR thus generates $2^{64} - 1$ different masks. It must not be again accessible by the user to prevent its reset. For this, it must be correctly implemented in hardware. We apply the mask by an XOR on the input of the 10-th round. This allows to prevent the adversary to get two encryption of the same plaintext, and therefore to make a DFA. For decryption, we apply an XOR between the mask and the output of the 10-th round and get the correct plaintext. We then have two options. The first is to send the mask with the ciphertext. Unfortunately in this case, this method does not protect against an attack on decryption. Indeed, the attacker can choose the same mask on each decryption. However, in the context of IoT it is common that the card is only used for encryption and decryption is carried out on a protected server. The second is to synchronize the encryption and the decryption. They both use the same LFSR with the same initialization and the decryption must be applied in the same order as ciphertexts received. Therefore, the countermeasure protects both the encryption and the decryption but with an additional synchronisation constraint.

Cost: The cost depends on the choice of the random mask generation. A simple LFSR - like the one we cited - implemented in hardware has a low cost with respect to IoT constraints. Moreover, applying the mask requests an additional cost of an XOR for encryption and the same for decryption in the second case.

7 Conclusion

In this paper we propose the first differential fault analysis on the block cipher PRIDE. We explain how this attack can be optimized and we demonstrate it, with 4 faults only to retrieve the full secret key. We show that our attack is indeed feasible from 32-bit random faults obtained with electromagnetic injection, which is a low-cost means of injection. We believe that the resistance against DFA is important for a cipher like PRIDE, which is expected to be largely deployed in low-end devices thanks to its lightness. At last we propose some countermeasures which leave the cipher still very efficient for IoT devices. They can be combined to provide more security and are not exhaustive. An optimization of these countermeasures is possible for make them less costly and keep the light side of the cipher. It is also necessary to be careful that the protections to prevent the DFA do not open doors to further attacks. Finally, it appears that our attack applies to any SPN-based block ciphers with a linear layer similar to the one used in PRIDE, like the LS-Designs family introduced by Grosso & al [19] in 2014. The details of this generalization will be studied in a future work.

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

References

1. Aciiçmez, O., Koç, Çetin Kaya., Seifert, J.P.: Predicting secret keys via branch prediction. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 225–242. Springer, Berlin, Germany, San Francisco, CA, USA (Feb 5–9, 2007)
2. Acimez, O., etin Kaya Ko, Seifert, J.P.: On the power of simple branch prediction analysis. In: 2007 ACM SYMPOSIUM ON INFORMATION, COMPUTER AND COMMUNICATIONS SECURITY (ASIACCS07. pp. 312–320. ACM Press (2007)
3. 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)
4. Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçın, T.: Block Ciphers – Focus on the Linear Layer (feat. PRIDE), pp. 57–76. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
5. Baysal, A., Sahin, S.: Roadrunner: A small and fast bitslice block cipher for low cost 8-bit processors. In: Güneysu, T., Leander, G., Moradi, A. (eds.) LightSec 2015. LNCS, vol. 9065, pp. 58–76. Springer, Berlin, Germany, Bochum, Germany (September 10-11, 2015)
6. 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
7. Bertoni, G., Zaccaria, V., Breveglieri, L., Monchiero, M., Palermo, G.: AES power attack based on induced cache miss and countermeasure. In: ITCC 2005, Volume 1. pp. 586–591. IEEE Computer Society, Las Vegas, Nevada, USA (April 4-5, 2005)
8. 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)

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. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Berlin, Germany, Cambridge, Massachusetts, USA (Aug 11–13, 2004)
14. Dai, Y., Chen, S.: Cryptanalysis of full PRIDE block cipher. Cryptology ePrint Archive, Report 2014/987 (2014), http://eprint.iacr.org/2014/987
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. Dhem, J., Koeune, F., Leroux, P., Mestré, P., Quisquater, J., Willems, J.: A practical implementation of the timing attack. In: Quisquater, J., Schneier, B. (eds.) CARDIS '98. LNCS, vol. 1820, pp. 167–182. Springer, Berlin, Germany, Louvain-la-Neuve, Belgium (September 14-16, 1998)
17. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Çetin Kaya., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Berlin, Germany, Paris, France (May 14–16, 2001)
18. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Berlin, Germany, Washington, D.C., USA (Aug 10–13, 2008)
19. 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)
20. 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
21. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 18–22, 1996)
22. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 15–19, 1999)

23. Koeune, F., Standaert, F.: A tutorial on physical security and side-channel attacks. In: Foundations of Security Analysis and Design III, FOSAD 2004/2005 Tutorial Lectures. LNCS, vol. 3655, pp. 78–108. Springer, Berlin, Germany (2005)
24. 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)
25. Maghrebi, H., Danger, J.L., Flament, F., Guilley, S., Sauvage, L.: Evaluation of countermeasure implementations based on Boolean masking to thwart side-channel attacks. In: International Signals, Circuits and Systems Conference - SCS 2009. pp. 1–6 (2009)
26. Maghrebi, H., Guilley, S., Danger, J., Flament, F.: Entropy-based power attack. In: Plusquellic, J., Mai, K. (eds.) HOST 2010. pp. 1–6. IEEE Computer Society, Anaheim Convention Center, California, USA (June 13-14, 2010)
27. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
28. 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)
29. Page, D.: Theoretical use of cache memory as a cryptanalytic side-channel. Cryptology ePrint Archive, Report 2002/169 (2002), http://eprint.iacr.org/2002/169
30. Page, D.: Defending against cache based side-channel attacks. Information Security Technical Report 8(1), 30–44 (April 2004)
31. Quisquater, J., Samyde, D.: Electromagnetic analysis (EMA): measures and countermeasures for smart cards. In: E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Berlin, Germany, Cannes, France (September 19-21, 2001)
32. 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)
33. Skorobogatov, S.: Semi-invasive attacks - A new approach to hardware security analysis. Technical Report 630, University of Cambridge (April 2005)
34. 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)
35. 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
36. 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
37. Yang, Q., Hu, L., Sun, S., Qiao, K., Song, L., Shan, J., Ma, X.: Improved differential analysis of block cipher PRIDE. In: Lopez, J., Wu, Y. (eds.) ISPEC 2015. LNCS, vol. 9065, pp. 209–219. Springer, Berlin, Germany, Beijing, China (May 5-8, 2015)
38. Zhao, J., Wang, X., Wang, M., Dong, X.: Differential analysis on block cipher PRIDE. Cryptology ePrint Archive, Report 2014/525 (2014), http://eprint.iacr.org/2014/525
39. 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 the PRIDE S-box

A.1 Difference distribution table of the PRIDE S-box

Table 12 shows the difference distribution table T of the PRIDE S-box which is defined by $T(i,j) = \#\{(x,y) \in \{0,1\}^4 \times \{0,1\}^4 \mid x \oplus y = i, \mathcal{S}(x) \oplus \mathcal{S}(y) = j\}$.

Table 12: difference distribution table of the PRIDE S-box

T	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7	0x8	0x9	0xa	0xb	0xc	0xd	0xe	0xf
0x0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x1	0	0	0	0	4	4	4	4	0	0	0	0	0	0	0	0
0x2	0	0	0	0	0	0	0	0	4	0	0	4	2	2	2	2
0x3	0	0	0	0	0	0	0	0	4	0	0	4	2	2	2	2
0x4	0	4	0	0	0	0	4	0	0	2	2	0	2	0	0	2
0x5	0	4	0	0	0	4	0	0	0	2	2	0	2	0	0	2
0x6	0	4	0	0	4	0	0	0	0	2	2	0	0	2	2	0
0x7	0	4	0	0	0	0	0	4	0	2	2	0	0	2	2	0
0x8	0	0	4	4	0	0	0	0	4	0	4	0	0	0	0	0
0x9	0	0	0	0	2	2	2	2	0	0	0	0	2	2	2	2
0xa	0	0	0	0	2	2	2	2	4	0	4	0	0	0	0	0
0xb	0	0	4	4	0	0	0	0	0	0	0	0	2	2	2	2
0xc	0	0	2	2	2	2	0	0	0	2	0	2	2	0	2	0
0xd	0	0	2	2	0	0	2	2	0	2	0	2	0	2	0	2
0xe	0	0	2	2	0	0	2	2	0	2	0	2	2	0	2	0
0xf	0	0	2	2	2	2	0	0	0	2	0	2	0	2	0	2

A.2 Proof of Proposition 1

We can see that, from the knowledge of a nonzero input $(x \oplus y)$ and of an output difference $(\mathcal{S}(x) \oplus \mathcal{S}(y))$ for $\mathcal{S}$ we deduce 0, 2 or 4 candidates for the input value x. Moreover, we can easily find pairs of differentials (a_1, b_1) and (a_2, b_2) which are satisfied by a single input x. For this, we use Proposition 1 that we prove here.

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

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

Let us first prove that both $\mathcal{D}(a_1, b_1)$ and $\mathcal{D}(a_2, b_2)$ have at least 4 elements. If these two sets have two elements only, $\mathcal{D}(a_1, b_1) = \{x, x \oplus a_1\}$ and $\mathcal{D}(a_2, b_2) = \{x, x \oplus a_2\}$, implying that they cannot be the same since $a_1 \neq a_2$. Then, at least one of the two sets contains at least four elements. Suppose that $\#\mathcal{D}(a_1, b_1) = 4$ and $\#\mathcal{D}(a_2, b_2) = 2$. Then, $x \oplus a_2 \in \mathcal{D}(a_1, b_1)$, with $\mathcal{D}(a_2, b_2) = \{x, x \oplus a_2\}$. Consequently,

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

implying that

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

Thus $x \oplus a_1 \in \mathcal{D}(a_2, b_2)$, a contradiction. We have proved that $\#\mathcal{D}(a_2, b_2) = 4$. Now, 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 $x \oplus a_2 \in \mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$.

Suppose now that $\{x, x \oplus a_4\} \subseteq \mathcal{D}(a_1, b_1) \cap \mathcal{D}(a_2, b_2)$ for some $a_4 \neq 0$, we deduce that the four elements $x \oplus a_1$, $x \oplus a_2$, $x \oplus a_1 \oplus a_4$ and $x \oplus a_2 \oplus a_4$ belong to $\mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2)$. These four elements are either distinct or satisfy $a_4 = a_1 \oplus a_2$ which implies that $x \oplus a_4 \oplus a_2 = x \oplus a_1$ 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 $\#\mathcal{D}(a_1 \oplus a_2, b_1 \oplus b_2) = 4$. □

B Other trade-offs between the number of faults and the time complexity

We have shown that 4 faults with an appropriate strategy enable the attacker to recover the whole key. In this section, we evaluate the number of key candidates that an attacker can obtain with fewer faults. This number then corresponds to the time complexity of the complete key recovery. Indeed, if the attacker knows a pair of plaintext-ciphertext, encrypting the known plaintext under each key candidate until the correct ciphertext is recovered leading to a complete key recovery[8]. Firstly, the number of remaining candidates for the subkey k_0 (resp. k_1) that an attacker can obtain with one fault on the 19-th round (resp. 18-th round) is 2^{32}. We now use this result to estimate the cost of the full key recovery from a few faults only.

With a single fault. We want to determine the cost of the key recovery if the attacker can inject a single fault. If this fault is injected in the 19-th round, then the possible values of k_0 is reduces to a list of 2^{32} candidates. This corresponds to a total of 2^{96} candidates for the whole 128-bit key. If the attacker knows two plaintext-ciphertext pairs, he can then encrypt the first known plaintext under each of these 2^{96} key candidates, until the corresponding ciphertext is recovered. Only $2^{96-64} = 2^{32}$ key candidates then remain, and the second plaintext-ciphertext pair can then be exploited for recovering the key. The main part of the time complexity in this attack is the cost of the exhaustive search over the 2^{96} candidates, which corresponds to 2^{96} encryptions.

If the fault is now injected in the 18-th round, then the attack consists in successively examining all 2^{64} possible values for k_0. For each of these 2^{64}

[8] provided that the number of key candidates is smaller than 2^{64}. Otherwise, two plaintext-ciphertext pairs are needed.

candidates, the attacker inverts the last encryption round for both the correct and the faulty ciphertexts C and C^*. He deduces the value of ΔX_{20}, and then of ΔY_{19}. When choosing a random k_0, ΔX_{20} varies in the set of all input differences which can appear when the output difference equals ΔY_{20}. From the difference distribution table of the S-box, the average number of valid input differences corresponding to a fixed output difference is

$$\frac{1}{16}(1 + 4 \times 2 + 6 \times 8 + 8 \times 5) = 6.0625.$$

Therefore, ΔX_{20} (and then ΔY_{19}) takes in average $6.0625^{16} = 2^{41.6}$ different values, and each of these differences appears for $2^{22.4}$ values of k_0 in average.

But, the difference ΔX_{20} is not valid if the corresponding value of ΔY_{19} does not have the form expected from the value of the fault. As the fault has been injected on Z^0_{18} or Z^3_{18}, each nibble of ΔX_{19} is equal either to 0x1, or to 0x8. Then, the corresponding nibble ΔY_{19} can take 4 values only. Therefore, the proportion of valid values for ΔY_{19} is $4^{16} \times 2^{-64} = 2^{-32}$. It follows that, among the $2^{41.6}$ values of ΔY_{19} which are obtained from the partial decryption, only $2^{9.6}$ are valid, implying that only 2^{32} values of k_0 need to be considered. For each of these 2^{32} values of k_0, the value of the fault, and then of ΔX_{19} provides 2^{32} candidates for k_1 as proved in the previous section. This step then leads to a list of 2^{64} candidates for the whole 128-bit key, with a time complexity which mainly corresponds to the cost for decrypting one round of PRIDE 2^{64} times. The bottleneck of the attack is then the final key recovery procedure, which consists in testing the 2^{64} remaining keys on two plaintext-ciphertext pairs. The overall cost of the attack is then roughly the cost of 2^{64} encryption.

With two faults. If the two faults are injected in the 18-th round, then the previously described technique which enables the attacker to eliminate some candidates for k_0 is repeated twice. Only a proportion of 2^{-64} values of ΔY_{19} will be valid, implying that only the correct value of ΔY_{19} will remain after this step. As previously explained, each value of ΔY_{19} is obtained for $2^{22.4}$ values of k_0 in average. Therefore, this sieving procedure leads to a list of $2^{22.4}$ candidates for k_0. Now, exploiting the two faults injected in the 18-th round provides one candidate for k_1. Therefore, we get $2^{22.4}$ candidates for the whole key. The total time complexity of the attack then corresponds to 2^{64} decryption of a single round, and to an exhaustive search among the $2^{22.4}$ remaining keys. The first step is then the bottleneck and its cost is less than the cost of $2^{64}/20 = 2^{59.7}$ complete encryptions.

If the first fault is now injected in the 19-th round, then the list of possible values for k_0 is first reduced to a list of size 2^{32} as explained in the previous section. The second fault, injected on the 18-th round, then enables to reduce this list to $2^{32-32} = 1$ possible value for k_0. For this value of k_0, a list of 2^{32} candidates for k_1 is obtained from the second fault. The number of candidates for the whole key, which need to tested, is then 2^{32}. The bottleneck of the attack is then the exhaustive search over the 2^{32} remaining key candidates, which corresponds to a time complexity equal to the cost of 2^{32} encryptions.

With three faults. The best strategy with three faults consists in injecting one fault in the 19-th round, and two in the 18-th round. From the fault in the 19-th round, the attacker gets a list of 2^{32} candidates for k_0. By decrypting the last round under these 2^{32} values of k_0, we roughly get 2^{32} pairs of values for ΔY_{19} among which one is expected to be consistent with the two faults injected in the 18-th round. Moreover, these two faults lead to one candidate for k_1, i.e., one candidate for the whole key. The time complexity of the attack then corresponds to the cost of 2^{32} encryptions of a single round, i.e., $2^{27.7}$ full encryptions.

C ARM source code

C.1 L-layer

; $\mathcal{L}_0$ and $\mathcal{L}_1$
; State s_0
; Temporary registers $t_0, \cdots, t_6$
(1) MOV t_0, #0x00F0
(2) MOVT t_0, #0xF0F0
(3) AND t_1, t_0, s_0, LSL#4
(4) LSR t_0, #4
(5) AND t_2, t_0, s_0, LSR#4
(6) AND t_0, s_0, #0xFF000000
(7) AND t_3, s_0, #0XFF0000
(8) EOR t_1, t_1, t_2
(9) AND s_0, s_0, #0xFF00
(10) EOR s_0, s_0, t_1
(11) AND t_1, s_0, #0x8000
(12) AND t_2, s_0, #0x01
(13) AND t_4, s_0, #0xFF00
(14) AND t_5, s_0, #0x00FF
(15) MOV t_6, #0xFF000000
(16) AND t_6, t_6, s_0, LSL#8
(17) EOR s_0, s_0, r10
(18) AND t_6, s_0, #0xFF000000
(19) EOR t_0, t_0, t_6
(20) BIC s_0, s_0, #0xFF0000
(21) EOR s_0, s_0, t_0, LSR#8
(22) EOR s_0, s_0, t_3, LSL#8
(23) MOV t_0, #0xFF00
(24) AND t_0, t_0, t_4, LSL#1
(25) EOR t_0, t_0, t_1, LSR#7
(26) LSR t_3, t_5, #1
(27) EOR t_3, t_3, t_2, LSL#7
(28) EOR s_0, s_0, t_3, LSL#8
(29) AND t_3, s_0, #0xFF00
(30) EOR s_0, s_0, t_0
(31) EOR s_0, s_0, t_3, LSR#8

; $\mathcal{L}_2$ and $\mathcal{L}_3$
; State s_1
; Temporary registers $t_0, \cdots, t_5$
(1) MOV t_0, #0xF0F0
(2) MOVT t_0, #0xF000
(3) AND t_1, t_0, s_1, LSL#4
(4) LSR t_0, #4
(5) AND t_2, t_0, s_1, LSR#4
(6) AND t_0, s_1, #0xFF00
(7) AND t_3, s_1, #0X00FF
(8) AND s_1, s_1, #0xFF0000
(9) EOR t_1, t_1, t_2
(10) EOR s_1, s_1, t_1
(11) AND t_1, s_1, #0x80000000
(12) AND t_2, s_1, #0x00010000
(13) MOV t_4, #0xFF000000
(14) AND t_5, s_1, t_4
(15) AND t_4, t_4, t_5, LSL#1
(16) EOR t_1, t_4, t_1, LSR#7
(17) MOV t_4, #0x00FF0000
(18) AND t_5, s_1, t_4
(19) AND t_4, t_4, t_5, LSR#1
(20) EOR t_2, t_4, t_2, LSL#7
(21) EOR s_1, s_1, t_2, LSL#8
(22) AND t_2, s_1, #0xFF000000
(23) EOR s_1, s_1, t_1
(24) EOR s_1, s_1, t_2, LSR#8
(25) AND t_4, s_1, #0x00FF
(26) EOR s_1, s_1, t_4, LSL#8
(27) AND t_4, s_1, #0xFF00
(28) EOR t_3, t_3, t_4, LSR#8
(29) EOR t_0, t_0, t_3
(30) BIC s_1, s_1, #0x00FF
(31) EOR s_1, s_1, t_0

C.2 S-layer

; State s_0, s_1
; Temporary registers t_0, t_1
(1) MOV t_1, s_0
(2) AND t_0, s_0, s_0, LSL#16
(3) EOR t_0, t_0, s_1
(4) AND s_0, s_0, s_1, LSR#16
(5) EOR s_0, s_0, t_0
(6) AND t_0, s_0, s_0, LSL#16
(7) EOR t_0, t_0, t_1
(8) AND s_1, s_0, t_0, LSR#16
(9) EOR s_1, s_1, t_0