# On the importance of considering physical attacks when implementing lightweight cryptography

Alexandre Adomnicai, Benjamin Lac, Anne Canteaut, Jacques Jean-Alain Fournier, Laurent Masson, Renaud Sirdey, Assia Tria

# On the importance of considering physical attacks when implementing lightweight cryptography

Alexandre Adomnicai[1,6], Benjamin Lac[2,6], Anne Canteaut[5], Jacques J.A. Fournier[3]
Laurent Masson[1], Renaud Sirdey[4], and Assia Tria[2]

[1] Trusted Objects, Rousset, France,
{a.adomnicai, l.masson}@trusted-objects.com
[2] CEA-Tech, Gardanne, France,
[3] CEA-Leti, Grenoble, France,
[4] CEA-List, Saclay, France,
{benjamin.lac, renaud.sirdey, assia.tria, jacques.fournier}@cea.fr
[5] Inria, Paris, France,
anne.canteaut@inria.fr
[6] ENSM-SE, Saint-Étienne, France,

**Abstract.** Pervasive devices are usually deployed in hostile environments where they are physically accessible to attackers. As lightweight cryptography is designed for such devices, it has to be particularly resistant to physical attacks. In this paper, we illustrate how active and passive physical attacks against the lightweight block cipher PRIDE can be carried. A side channel attack and a fault attack have been successfully implemented on the same software implementation of the algorithm. In both cases, we were able to recover the entire encryption key. First, we present our attacks, then we analyze them in terms of complexity and feasibility and finally, we discuss possible countermeasures.

**Keywords:** LWC · PRIDE · Physical attacks · CEMA · DFA.

## 1 Introduction

Everyday, more objects are turned into interconnected pervasive devices. The expansion of the Internet of Things (IoT) brings many benefits but also raises a number of problems concerning security and privacy. Security is one of the biggest barriers to IoT adoption. To tackle this challenge, lightweight cryptography (LWC) is investigated in order to address IoT security issues while seeking the best compromise between security, power consumption, high performance and low footprint. During the last years, several lightweight block ciphers have been proposed, for example PRIDE [3], PRESENT [9], CLEFIA [33], PRINCE [11], KLEIN [15], SIMON [5] or SPECK [5]. LWC will be embedded into the IoT devices which shall have to store and handle secret/sensitive cryptographic keys at some points. The security of these keys within the device has to be guaranteed throughout the life cycle of the device (*i.e.* from the device's manufacturing through the personalization stage up to its end of life), which may last several years. In the meantime, the device will be in the field and as it can be a hostile environment (*i.e.* physically accessible to attackers), physical attacks must be taken in account. 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 [7], PICARO [26], Zorro [17] and the LS-designs family [18]. Although hardware implementations are more efficient in all aspects (performances, power consumption and security) than software ones, design and study of software-oriented ciphers is nevertheless important since these implementations are widely used in practice because of their flexibility and ease of development. In this paper we analyze the resistance of PRIDE against physical attacks because nowadays, when looking at software implementations, it is one of the most efficient lightweight block ciphers [4] as shown by the performance comparisons given in [3,4]. In this paper we first present the PRIDE algorithm before introducing physical attacks. Then we introduce the two attacks that have been put into practice before analyzing them in terms of efficiency and feasibility. Finally we discuss countermeasures that can be implemented to thwart such attacks before concluding the paper with some perspectives.

## 2 The PRIDE block cipher

PRIDE is an iterative block cipher composed of 20 rounds and introduced by Albrecht & al. [3] 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 algorithm to produce the subkeys $f_r(k_1)$ for each round $r$. The key schedule adds round-constants to parts of the key.

We denote by $k_{1_i}$ 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

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

The design of PRIDE is close to the one of a LS-design, a concept that was introduced by Grosso & al [18] in 2014, the only differences being that it uses an additional key for pre- and post-whitening, several matrices for the linear layer and has no linear layer on the last round. In this paper, we chose to present PRIDE as a LS-design in order to explain more simply our analysis.The inner state of the cipher, as well as the plaintext, ciphertext, and key, are all represented as a $4 \times 16$ bits array. In this paper, $B[n]$ denotes the $n$-th nibble (4 bits) of a binary word $B$ while $B_n$ denotes the $i$-th byte of $B$. Moreover, the nibbles' rows and columns are numbered from left to right starting from 1. The following notations are used for the intermediate values of the state within a round function:

| | |
|---|---|
| $I_r$ | 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 |
| $O_r$ | the output of the $r$-th round |

A round $r$ such that $1 \leq r \leq 19$ is composed of the following steps:

i. XORing the current $n$-bit subkey $f_r(k_1)$ with the state: $X_r = I_r \oplus f_r(k_1)$,

ii. Applying the 4-bit S-box $\mathcal{S}$, which definition is given in Appendix C, to each column of the state (*i.e.* apply the substitution layer $\mathcal{S}-\mathsf{layer}$ to the state): $Y_r = \mathcal{S}-\mathsf{layer}(X_r)$,

iii. Multiplying each row by a matrice $\mathcal{L}_i$, called L-box, given in [3] for $0 \leq i \leq 3$ (*i.e.* apply the linear layer $\mathcal{L}-\mathsf{layer}$ to the state): $O_r = \mathcal{L}-\mathsf{layer}(Y_r)$.

The last round simply consists of the first two steps (*i.e.* without the linear layer).

In order to encrypt a plaintext $M$, the cipher performs a XOR between $M$ and $\mathcal{P}(k_0)$, where $\mathcal{P}$ is the permutation layer given in [3]. It then applies the 20 rounds as previously described, and finally applies once again a XOR between $M$ and $\mathcal{P}(k_0)$. Figure 1 shows the representation of PRIDE inner state with frames showing the inputs of S-box and the input of L-box.

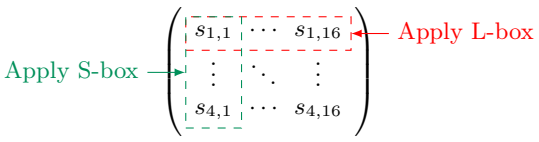


Figure 1: Inner matrix state of PRIDE

In this paper, we denote by $S_1 \cdots S_{16}$ the inner state given in Figure 1 such that $S_i$ consists of the nibble $s_{1,i} \cdots s_{4,i}$ for all $i$. For example, the hexadecimal value `0xe8d3157f246e80cb` denotes the inner state given in Figure 2.

$$\begin{pmatrix} 1&1&1&0&1&0&0&0&1&1&0&1&0&0&1&1 \\ 0&0&0&1&0&1&0&1&0&1&1&1&1&1&1&1 \\ 0&0&1&0&0&1&0&0&0&1&1&0&1&1&1&0 \\ 1&0&0&0&0&0&0&0&1&1&0&0&1&0&1&1 \end{pmatrix}$$

Figure 2: Inner state `0xe8d3157f246e80cb`

## 3 Physical attacks

Cryptographic algorithms are usually constructed to resist to algebraic (mathematical) cryptanalysis or exhaustive key searching by future computers. However, most cryptographic models do not cover physical attacks which target the cryptographic primitive's implementation. Physical attacks can be divided into two classes: passive attacks and active ones. Active attacks disturb the operation of a device or try to reverse-engineer functions by analysing the chip at the logic level. Passive attacks, also called side channel attack (SCA) [22], can be divided into timing attacks [14], and interpretation of one or more traces [28,24] (*i.e.* recording of the power or electromagnetic emanation while a cryptographic primitive is running on the device). In this paper, we present an attack from each category (passive and active) on the PRIDE lightweight block cipher.

### 3.1 Side-channel attacks

Since the publication of differential power analysis (DPA) [21], it is public knowledge that the analysis of a power trace obtained when executing a cryptographic primitive might reveal information about the secret involved.
A few years later, correlation power analysis (CPA) has been widely adopted over DPA as it requires fewer traces and is more efficient [12]. The principle is to recover part of the secret key by targeting a specific intermediate state of the algorithm, and try to predict its value by making hypotheses on the portion of the key involved. Then, to uncover the link between the predictions and the traces, the Pearson correlation coefficient between these two variables is computed using an appropriate leakage model (usually based on the Hamming weight or the Hamming distance depending on the platform and the targeted implementation). It yields a value between $-1$ (total negative correlation) and $+1$ (total positive correlation) for every point in time, indicating how much the prediction correlates to the recorded values over several traces. The formula of this coefficient is

$$\rho(X,\ Y) = \mathsf{Corr}(X,\ Y) = \frac{\mathsf{Cov}(X,\ Y)}{\sigma_X\ \sigma_Y} = \frac{\mathsf{E}(X\ Y) - \mathsf{E}(X)\mathsf{E}(Y)}{\sqrt{\mathsf{E}\Big(\big(X - \mathsf{E}(X)\big)^2\Big)\ \mathsf{E}\Big(\big(Y - \mathsf{E}(Y)\big)^2\Big)}}. \tag{1}$$

where $\mathsf{E}(X)$ is the expected value of the random variable $X$. Then, the hypothesis which matches with the real key should return a significantly higher coefficient than the other hypotheses. Note that other functions may be used to exploit the correlation between measured traces and the secret key used, like those based on template analyses or mutual information exploitation.

The attack described above remains valid when analyzing electromagnetic (EM) emanation traces instead of power consumption ones. In this case, we talk about Correlation-based ElectroMagnetic Analysis (CEMA). Although there are many different ways to measure EM emissions (sensor types, positioning...), this side channel has properties that make it more interesting than the "traditional" power consumption measurements. Among those properties, the ability to measure locally and in a contactless manner [2] makes electromagnetic emanations very attractive. Furthermore, power analysis often requires a slight modification of a device's printed circuit board (PCB) (*e.g.* by setting up a point to monitor core voltage), which is not necessary with EM analysis. These reasons led us to choose this side channel.

Regarding LWC, side channel attacks have been performed againsts ciphers like PRESENT [39,27,30], CLEFIA [29] or PRINCE and RECTANGLE [32].

### 3.2 Fault attacks

Fault attacks, introduced in [10], consist in disturbing the behaviour of the circuit in order to alter the correct execution of the cipher. Faults can be injected into the device by various means such as light pulses [35], laser [34], clock glitches [1], spikes on the voltage supply [8] or EM perturbations [13]. Some other techniques are not invasive, i.e. glitches (power, clock, electromagnetic). Clock and voltage glitches disturb the whole component while EM glitches allow to have more local effects with relatively high spatial and temporal precisions, using equipment at "affordable costs" [13].

One of the objectives of fault attacks, especially when considering cryptographic ciphers, is to perform Differential Fault Analysis (DFA). DFA, originally described in [6], consists in retrieving a cryptographic key by comparing the correct ciphertexts with the faulty ones yielded by computations during which a physical perturbation was applied. In the particular field of LWC, differential fault attacks have been proposed against ciphers like PRESENT [40], SPECK [38], TRIVIUM [25], PRINCE [36] and PRIDE [23]. DFA techniques are very efficient to retrieve keys used during a cryptographic computation, usually requiring only a few executions. For this main reason, in our analysis of PRIDE implementations security, we decided to first focus on its resistance against fault attacks in order to identify possible attack paths.

## 4 Setting up the attacks

In this section, we first describe how our device has been programmed and then detail the 'reverse-engineering' done to carry out the attacks presented afterwards.

### 4.1 Implementation

In order to test the feasibility of our attacks against PRIDE, we have implemented and run the cipher on a chip embedding an ARM Cortex-M3 micro-controller. That specific 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 attacks presented in this paper. Our implementation, whose source code is given in Appendix D, works on bytes of data. In our experiment, we used a key $k = k_0||k_1$ where $k_0 =$ `0xa371b246f90cf582` and $k_1 =$ `0xe417d148e239ca5d`.

### 4.2 Simple Electromagnetic Analysis

First, we performed a simple electromagnetic analysis (SEMA) during one execution of PRIDE in order to identify our attack targets.

By inspecting the trace shown in Figure 3, we could clearly distinguish every twenty rounds.

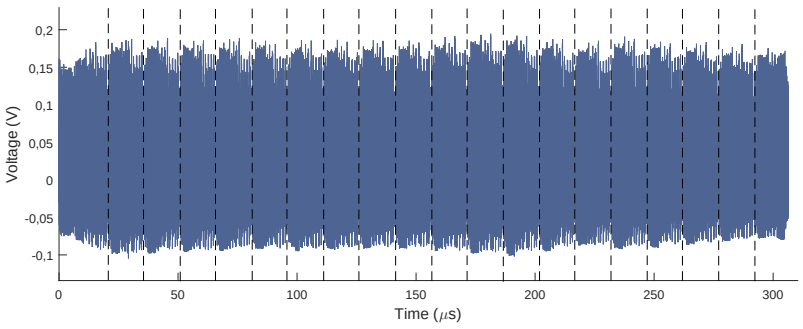


Figure 3: Electromagnetic emanations of the whole PRIDE block cipher

Then, we tried to recognize each operation by zooming onto the first two rounds, the corresponding trace is shown in Figure 4.

At first sight, it was easy to differentiate one round from the other but not one operation from the other. To distinguish each operation whithin a round, we first took a look at the last one, where the $\mathcal{L}-$layer is omitted. Consequently, it allowed us to determine the pattern corresponding to the $\mathcal{L}-$layer and so where the $\mathcal{S}-$layer's one ends. Finally, we had to distinguish the round key whitening and the $\mathcal{S}-$layer. As the round key whitening consist of 4 additions and 8 XORs, we made the educated guess depicted in Figure 4 intuitively.

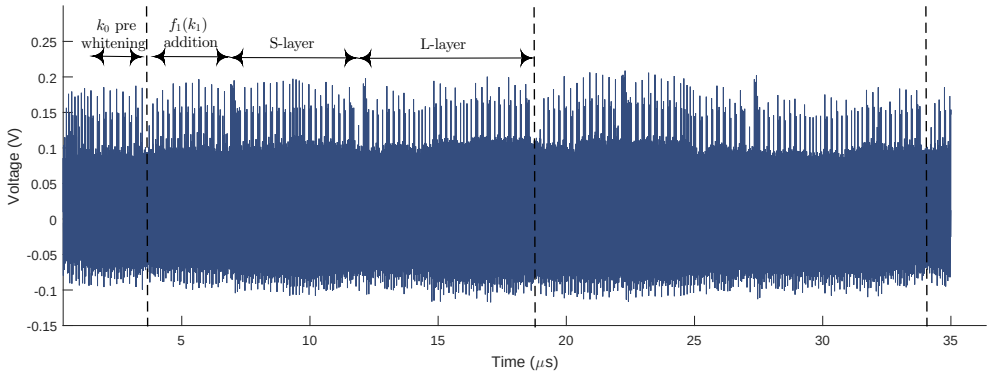


Figure 4: Electromagnetic emanations of the first two rounds of PRIDE block cipher

## 5 Correlation electromagnetic analysis

In this section, we introduce our attack to retrieve the secret key using CEMA on unprotected PRIDE computations. Then, we propose a pragmatic execution of the attack on our 8-bit implementation.

### 5.1 General principle

The principle is to make the attack in two stages: one for each halves of the key. The first step consists in recovering $\mathcal{P}(k_0)$. To do this, we chose to focus on the last round, as in the first round, $\mathcal{P}(k_0)$ and $f_1(k_1)$ are added successively to the state.

By characterizing our chip embedding an ARM Cortex-M3, we observed that information leaked upon register updates through the `STRB` ARM instruction. As the leak does not concern the previous state value, we used a leakage model based on the Hamming weight [7] (HW) of the manipulated data.

CEMA against block ciphers usually focuses on the input (or output, depending on whether the attack focuses on the last round or not) of the S-box operation which is the only non-linear element of the algorithm. This non-linearity ensures a good distinguishability between the correct and incorrect key guesses for CEMA. Indeed, correlation between the observed and the predicted EM leakage will be close to zero if the key guess is incorrect, due to the non-linear relationship between the predicted state and the key. Although we could focus on the input of the last $\mathcal{S}-$layer by starting from ciphertexts, we did not opt for this approach. At first glance, it seemed too convoluted because of our bitsliced implementation. It is due to the fact the permutations $\mathcal{P}$ and $\mathcal{P}^{-1}$ form an integral part of the $\mathcal{S}-$layer and have not been explicitly implemented. Therefore, to recover the state's first byte at the last $\mathcal{S}-$layer input, one should make hypotheses on $\mathcal{P}(k_0)_0$, $\mathcal{P}(k_0)_2$ and $\mathcal{P}(k_0)_4$ (*i.e.* on 24 bits). Contrary to some other block ciphers such as AES, where each byte passes through the S-box independently, in the case of PRIDE each byte depends on several others during the $\mathcal{S}-$layer operation. Consequently, we decided to focus on the key additions where each byte could be treated independently. The first stage consists in recovering $\mathcal{P}(k_0)$ by predicting the state value at the $\mathcal{S}-$layer output while the second one consists in recovering $f_{20}(k_1)$ by predicting the state value at the $\mathcal{L}-$layer output.

[7] The Hamming weight correponds to the number of ones in the binary representation of the data.

### 5.2 Practical implementation

PRIDE was executed for 1000 random plaintexts with the fixed key $k$ stated in the previous section. The last two rounds were targeted for the data acquisition and EM traces were captured with 6500 points per encryption of 1000 samples. Thereafter, we will note the matrix of traces.

$$T = \begin{bmatrix} T_0 \\ T_2 \\ \vdots \\ T_{6499} \end{bmatrix} = \begin{bmatrix} t_{0,0} & t_{1,1} & \cdots & t_{0,999} \\ t_{2,0} & t_{2,1} & \cdots & t_{2,999} \\ \vdots & \vdots & \ddots & \vdots \\ t_{6499,1} & t_{6499,2} & \cdots & t_{6499,999} \end{bmatrix}. \tag{2}$$

To recover each byte $\mathcal{P}(k_0)_i$ for $0 \leq i \leq 7$, we first computed the estimation matrices $E^i$ by computing the Hamming weight of each ciphertext $C_j$ for $0 \leq j \leq 999$ XORed with each key hypothesis $0 \leq H_K \leq 255$.

$$E^i = \begin{bmatrix} E^i_0 \\ E^i_1 \\ \vdots \\ E^i_{255} \end{bmatrix} = \begin{bmatrix} e^i_{0,0} & e^i_{0,1} & \cdots & e^i_{0,999} \\ e^i_{1,0} & e^i_{1,1} & \cdots & e^i_{1,999} \\ \vdots & \vdots & \ddots & \vdots \\ e^i_{255,0} & e^i_{255,1} & \cdots & e^i_{255,999} \end{bmatrix} \tag{3}$$

where $e^i_{H_K,j} = HW(C_{j,i} \oplus H_K)$.

Then, we performed a classical CEMA attack (also called *Vertical*) by computing the correlation coefficients matrices $P^i$ from $E^i$ and $T'$ where $T' \subset T$ denotes the traces points corresponding to the last $\mathcal{S}-$layer.

$$P^i = \begin{bmatrix} P^i_0 \\ P^i_1 \\ \vdots \\ P^i_{n-1} \end{bmatrix} = \begin{bmatrix} \rho^i_{0,0} & \rho^i_{0,1} & \cdots & \rho^i_{0,255} \\ \rho^i_{1,0} & \rho^i_{1,1} & \cdots & \rho^i_{1,255} \\ \vdots & \vdots & \ddots & \vdots \\ \rho^i_{n-1,0} & \rho^i_{n-1,1} & \cdots & \rho^i_{n-1,255} \end{bmatrix} \tag{4}$$

where $\#T' = n \approx 1300$ and $\rho^i_{t,H_K} = \mathsf{Corr}(T'_t, E^i_{H_K})$.

Figure 5 shows the plot corresponding to $P^1$.

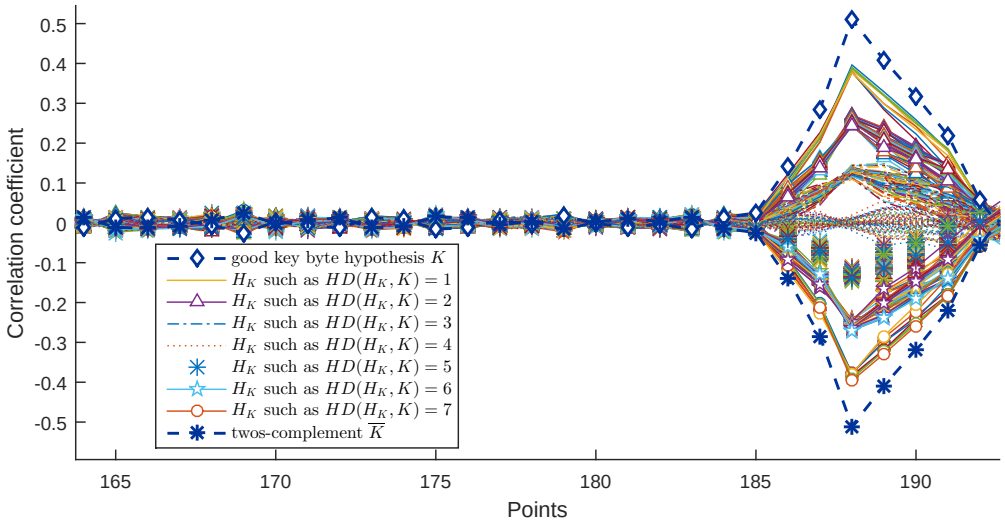


Figure 5: Key recovery of $\mathcal{P}(k_0)_0$ with 256-bit hypotheses

We can clearly distinguish a symmetry about the x-axis, which occurs due to the fact that the key hypotheses are simply XORed with the ciphertexts. Thus, the two's complement $\overline{H_K}$ (*i.e.* $255 - H_K$) of each key byte hypothesis $H_K$ leads to a symmetric relation regarding the Hamming weight estimation matrix $\big($*i.e.* $\forall i\ \forall j,\ E^i_{\overline{H_K},j} = 8 - E^i_{H_K,j}\big)$. This results in a negative correlation coefficient as stated in Proposition 1. The proof of this proposition is given in Appendix B.

**Proposition 1** *Let $X$ be an arbitrary variable. Let $Y_i = (y_{i,1}, y_{i,2}, ..., y_{i,n})$ and $Y_j = (y_{j,1}, y_{j,2}, ..., y_{j,n})$ be two variables such as $Y_i = z - Y_j$ i.e. $\forall n,\ y_{i,n} = z - y_{j,n}$ with $z \in \mathbb{R}$. Then,* $\mathsf{Corr}(X, Y_i) = -\mathsf{Corr}(X, Y_j)$.

Furthermore, we can differentiate 8 correlation classes. Each class corresponds to a set of key byte hypotheses $\mathcal{S}_d$ where the Hamming distance between the real key byte and each element equals $d$ (*i.e.* $\forall H_K \in \mathcal{S}_d,\ HD(H_K, K) = d$).

Therefore, we deduced that it was sufficient to make key byte hypotheses on 7 bits instead of 8. Consequently, in that way, if $\max(|P^i|) = \max(P^i)$ then the correct key byte is the matching $H_K$, otherwise it is $\overline{H_K}$. Figure 6 shows the plot corresponding to $P^1$ with 128-bit hypotheses and Figure 7 shows the plot corresponding to $P^2$ as well with 128-bit hypotheses which illustrates the other case (*i.e.* highest negative correlation coefficient).

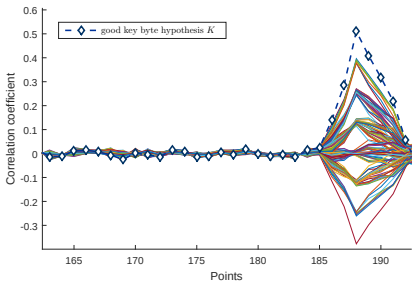


Figure 6: Key recovery of $\mathcal{P}(k_0)_0$ with 128-bit key hypotheses

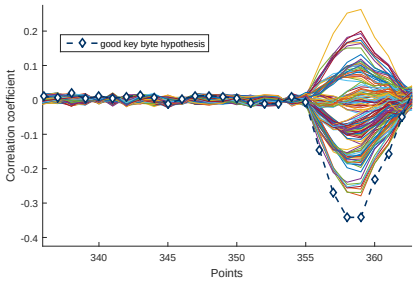


Figure 7: Key recovery of $\mathcal{P}(k_0)_1$ with 128-bit key hypotheses

In the same way, we were able to recover all the other bytes of $\mathcal{P}(k_0)$.
After that, we were able to apply the $\mathcal{S}-\mathsf{layer}$ without any complications and then we repeated the same reasoning to recover $f_{20}(k_1)$. The only differences concern the part of the trace which is analyzed $T'' \subset T$ (*i.e.* the $\mathcal{L}-\mathsf{layer}$ operation instead of the $\mathcal{S}-\mathsf{layer}$ one) and the way to compute the estimation matrices: $e^i_{H_K,j} = HW(C'_{j,i} \oplus H_K)$ where $C'_j = \mathcal{S}-\mathsf{layer}^{-1}\big(C_j \oplus \mathcal{P}(k_0)\big)$.

## 6 Differential faults analysis of PRIDE

In this section, we briefly recall the technique proposed in [23] to retrieve the secret key using fault injections on PRIDE computations. More details on this attack are given in [23]. Then, we propose a practical implementation of the attack on our 8-bit PRIDE implementation. Note that in this section, we chosed to apply $\mathcal{P}^{-1}$ to the differential inputs (resp. outputs) to clearly exhibit each S-box nibble input (resp. output).

### 6.1 General principle

In the first state of this attack, we corrupt, one by one, some rows of the inner state between the last two substitution layers in order to retrieve $k_0$. Indeed, a flip of the bit $1 \leq \alpha \leq 16$ on the row $1 \leq \beta \leq 4$ of $X_r$ at round $1 \leq r \leq 20$ gives us a difference $\Delta In_r[\alpha]$ equals to $2^{4-\beta}$ on the S-box input $\alpha$. Moreover, from the knowledge of the correct and the faulty ciphertexts $C$ and $C^*$, we can compute the corresponding difference $\Delta Out_r[\alpha]$ on the S-box output. Thereby, we obtain a known differential $(\Delta In_r[\alpha], \Delta Out_r[\alpha])$.

The best case consists then in flipping all the bits of the row in order to activate all the S-boxes in the last round. For example Figure 8 shows the obtained state difference from a flip of the second row before the substitution layer. In this case, we got a difference equal to 0x4 on the input of each S-box.

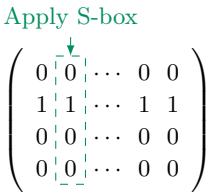


Figure 8: State difference obtained from a flip of the second row before the substitution layer

Then, we exploit the difference distribution table of the PRIDE S-box given in [23]. Indeed, obtaining information on $k_0$ is possible from the following equation on each nibble $1 \leq \alpha \leq 16$:

$$\Delta In_{20}[\alpha] = \mathcal{S}\big((\mathcal{P}^{-1}(C) \oplus k_0)[\alpha]\big) \oplus \mathcal{S}\big((\mathcal{P}^{-1}(C^*) \oplus k_0)[\alpha]\big)$$

Indeed,

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

$$x \oplus y = \Delta Out_{20}[\alpha] \text{ and } \mathcal{S}(x) \oplus \mathcal{S}(y) = \Delta In_{20}[\alpha]$$

and, from the knowledge of a nonzero input difference $\Delta In_{20}[\alpha]$ and of an output difference $\Delta Out_{20}[\alpha]$ for $\mathcal{S}$, we deduce 2 or 4 candidates for the input value $x$ because the differential uniformity of $\mathcal{S}$ equals 4 (as we can see from the difference distribution table of the PRIDE S-box). Moreover, Proposition 2 introduced in [23] enables us to exhibit pairs of differentials for the S-box which are simultaneously satisfied for a single element. The proof of this proposition is given in [23].

**Proposition 2** *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*

$$\mathcal{S}(x \oplus a_1) \oplus \mathcal{S}(x) = b_1 \tag{5}$$

$$\mathcal{S}(x \oplus a_2) \oplus \mathcal{S}(x) = b_2 \tag{6}$$

*has at least two solutions. Then, each of the three equations (5), (6) 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 (5) and (6) does not have any solution either.*

Finally, for the first stage (which objective is to find $k_0$), we just have to flip two rows such that the obtained pairs of differentials complies with the proposition. For example, flipping the first row and then the last one, allows us to obtain respectively for all $1 \leq \alpha \leq 16$ pairs of differentials $(\Delta Out_{20}[\alpha], \Delta In_{20}[\alpha])_1 = (a_1, \texttt{0x8})$ and $(\Delta Out_{20}[\alpha], \Delta In_{20}[\alpha])_2 = (a_2, \texttt{0x1})$ with $a_1$ and $a_2$ known. Since $\texttt{0x8} \oplus \texttt{0x1} = \texttt{0x9}$ from the Proposition 2 (and the difference distribution table of the PRIDE S-box), only one element in the intersection of the two sets of solutions is obtained for each nibble. Therefore, we have shown that we get only one candidate for each nibble of $\mathcal{P}^{-1}(C) \oplus k_0$ and, from the knowledge of $C$, we retrieve $k_0$.

Once $k_0$ has been recovered, $X_{20}$ and $X^*_{20}$ can be computed from the ciphertexts $C$ and $C^*$. Then $\Delta Out_{19}$ can be computed and the following equation, on each nibble $1 \leq \alpha \leq 16$,

$$\begin{aligned}\Delta In_{19}[\alpha] = &\Big(\mathcal{S} \circ \mathcal{P}^{-1} \circ \mathcal{L}\text{−layer}^{-1}\Big)\Big(\mathcal{S}\text{−layer}\big(C \oplus \mathcal{P}(k_0)\big) \oplus f_{20}(k_1)\Big)[\alpha] \\ &\oplus\Big(\mathcal{S} \circ \mathcal{P}^{-1} \circ \mathcal{L}\text{−layer}^{-1}\Big)\Big(\mathcal{S}\text{−layer}\big(C^* \oplus \mathcal{P}(k_0)\big) \oplus f_{20}(k_1)\Big)[\alpha],\end{aligned}$$

allows the attacker to recover $f_{20}(k_1)$, and therefore $k_1$ with the same method but from fault injections between the penultimate two substitution layers. Indeed,

$$\begin{aligned}x &= \Big(\mathcal{P}^{-1} \circ \mathcal{L}\text{−layer}^{-1}\Big)\Big(\mathcal{S}\text{−layer}\big(C \oplus \mathcal{P}(k_0)\big) \oplus f_{20}(k_1)\Big)[\alpha] \text{ and} \\ y &= \Big(\mathcal{P}^{-1} \circ \mathcal{L}\text{−layer}^{-1}\Big)\Big(\mathcal{S}\text{−layer}\big(C^* \oplus \mathcal{P}(k_0)\big) \oplus f_{20}(k_1)\Big)[\alpha] \text{ satisfy} \\ x \oplus y &= \Delta Out_{19}[\alpha] \text{ and } \mathcal{S}(x) \oplus \mathcal{S}(y) = \Delta In_{19}[\alpha].\end{aligned}$$

### 6.2 Practical implementation

In order 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 were able to inject faults at precise spatial locations and at precise enough instants to target specific rounds of the cipher during its execution. The set-up we used is quite similar to the one described in [13] but we did not need any motorized X-Y stage: injecting faults 'in the center' of the chip was good enough to have a random fault model (one chance out of two to flip a bit). Indeed, as we saw in the previous section, it is possible to target a precise 8-bit word (more precisely a specific instruction) but the injected faults follow a random pattern. We can thus retrieve the value of a random fault from the position of active S-boxes.

The plaintext used for all executions was `0xe8d3157f246e80cb` and the correct ciphertext was `0x0b735baaf63aac9e`. We used the SEMA presented in figure 3 to identify the last rounds in time. Then, we used an electromagnetic pulse generator to disrupt the PRIDE execution. All the obtained faults which were exploitable are given in Appendix A. Among the obtained faults, we underline in Appendix A those that give as much information as all faults. It also lists the candidates that we can extract from them.

From the obtained faults on the last two substitution layers and from $\mathcal{P}^{-1}(C) =$ `0x3636d3ec58eb71f8`, with $k_0[3] \in \{$`0x0`, `0x1`, `0x4`, `0x5`$\}$ and $k_0[11] \in \{$`0x8`, `0x9`, `0xc`, `0xd`$\}$, we got 16 possible values for $k_0$. In order to reduce the number of possible keys, we then used faulty ciphertexts obtained from fault injection between the penultimate two substitution layers. For this, we computed the difference output $\Delta Out_{19}$ from all the 16 remaining candidates for the key. Then, we observed that some differentials $(\Delta Out_{19}, \Delta In_{19})$ were not possible on the inverse S-box and therefore we removed the corresponding candidates.

Indeed, from the faulty ciphertext `0xc42ec0dbb65e18db`, we obtained the 16 following values for $\Delta Out_{19}$ for each possible value of $k_0$:

| $k_0$ | $\Delta Out_{19}$ | $k_0$ | $\Delta Out_{19}$ |
|---|---|---|---|
| 0xa370b246f908f582 | 0x000800096445640e | 0xa374b246f908f582 | 0x000800096445640e |
| 0xa370b246f909f582 | 0x020800096447440e | 0xa374b246f909f582 | 0x020800096447440e |
| 0xa370b246f90cf582 | 0x0000000064446407 | 0xa374b246f90cf582 | 0x0000000064446407 |
| 0xa370b246f90df582 | 0x0000000164456406 | 0xa374b246f90df582 | 0x0000000164456406 |
| 0xa371b246f908f582 | 0x000800096445640e | 0xa375b246f908f582 | 0x000800096445640e |
| 0xa371b246f909f582 | 0x020800096447440e | 0xa375b246f909f582 | 0x020800096447440e |
| 0xa371b246f90cf582 | 0x0000000064446407 | 0xa375b246f90cf582 | 0x0000000064446407 |
| 0xa371b246f90df582 | 0x0000000164456406 | 0xa375b246f90df582 | 0x0000000164456406 |

and as we knew that we injected faults on the last row of $X_{19}$, we knew that each nibble of $\Delta In_{19}$ was either `0x0` or `0x1`. From the difference distribution table of the S-box, we saw that an input difference equals to `0x1` implies an output difference in {`0x4`, `0x5`, `0x6`, 0x7}. Then, we got only four possible candidates for $k_0$ (displayed in red). Similarly, from the faulty ciphertext `0x3165d7eea5f5f4dc`, we obtained the following values for $\Delta Out_{19}$ based on remaining values of $k_0$:

| $k_0$ | $\Delta Out_{19}$ |
|---|---|
| 0xa370b246f90cf582 | 0x03a98a8300000001 |
| 0xa371b246f90cf582 | 0x03a88a8200000000 |
| 0xa374b246f90cf582 | 0x03a9ce8b40080009 |
| 0xa375b246f90cf582 | 0x21a9ce8b40082009 |

and as we knew that the fault was injected on the first row of $X_{19}$, we were able to retrieve $k_0$ (displayed in red).

Then, by doing the intersection between the sets for each nibble obtained from the faults injected between the penultimate two substitution layers, we got

$$\Big(\mathcal{P}^{-1} \circ \mathcal{L}\text{−layer}^{-1}\Big)\Big(\mathcal{S}\text{−layer}\big(C \oplus \mathcal{P}(k_0)\big) \oplus f_{20}(k_1)\Big) = \texttt{0xdf36eb60a400d4e9}.$$

Thus, $\mathcal{S}\text{−layer}\big(C \oplus \mathcal{P}(k_0)\big) \oplus f_{20}(k_1) = \texttt{0xffb81d4c69243ad7}$, and from

$$\mathcal{S}\text{−layer}\big(C \oplus \mathcal{P}(k_0)\big) = \texttt{0x1b93cc608ba9f016},$$

we deduced $f_{20}(k_1) = \texttt{0xe42bd12ce28dcac1}$ and finally, we retrieved $k_1 = \texttt{0xe417d148e239ca5d}$.

## 7 Costs analysis of CEMA and DFA on PRIDE

### 7.1 Attack paths

In terms of attack paths, CEMA exploits the key addition layer while DFA exploits the design of the PRIDE $\mathcal{S}$−layer. This latter makes the CEMA more tricky since it uses the transparent bitwise permutation layers given in [3] unlike classical substitution layers, which apply $n$-bit S-boxes singly on each $n$-bit words of the state. In the case of PRIDE, the substitution layer applies bitwise mathematical operations between each 16-bit words of the state (or bytes in our implementation). Consequently, it makes the intermediate state corresponding to input (or output) of the $\mathcal{S}$−layer more delicate to target. However, attacking a simple XOR operation still allowed us to carry out the attack.

On the other hand, this property makes DFA much easier. Indeed, flipping the 16 bits of any row at its input activates all S-boxes in the next round. Hence, applying this property in the last round allows the attacker to recover information on all nibbles of the subkey $k_0$. Then, the number of remaining candidates for $k_0$ is upper-bounded by $\delta(\mathcal{S})^{16}$, where $\delta(\mathcal{S}) = 4$ is the differential-uniformity of the PRIDE S-box. Moreover, the differential properties of the S-box 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 16 consecutive bits before the substitution layer, are enough to obtain a single candidate for the subkey.

### 7.2 Costs

We now analyze the total cost of each attack. First, we study the attacks implementation cost. CEMA only requires many curves of simple electromagnetic analysis of the last rounds from different plaintexts. In this case the ring oscillator does not need to be particularly efficient and a simple picoscope would amply do the job. DFA is more difficult to implement: it only needs one simple electromagnetic analysis but requires an electromagnetic pulse generator. The number of needed pulses in order to obtain enough exploitable faults is close to the number of required curves for the CEMA but DFA requires only one plaintext. Then, we compute an approximation of each attack complexity from the required parameters.

In case of CEMA, we have shown that, by attacking the key addition layer, it is sufficient to make hypotheses on 7 bits only. So, for each half of the key, we have to make $2^7 \times 8 = 2^{10}$ hypotheses. It means that our attack reduces the key search space from $2^{128}$ to $2^{11}$. To generalize, we denote $n_K$ the number of portion key hypotheses, $n_T$ the number of texts and $n_P$ the number of points per trace. Then, CEMA requires to compute $n_K \times n_C$ estimations and $n_K \times n_P$ correlation coefficients for each part of

the key. Note that the attack can be optimized by reducing the number of points treated. For example, an educated guess on the interval to attack can be made in order to avoid computations overhead. This underlines that CEMA requires much more memory than DFA. In this experimentation, approximately 100Mo were required but depending on $n_K$, $n_T$ and $n_C$ values, it can quickly become a handicap.

In case of DFA, we compute the number of remaining candidates from 8-bit random faults. We call 8-bit random fault the fact of having one chance out of two to flip each bit of a byte. This is close to what we have obtained in practice with electromagnetic pulses on our implementation. It is possible to target a precise word (more precisely a specific instruction) but the injected faults follow a random pattern. Moreover, injecting the faults before the linear layer allows us to obtain a difference pattern close to a 16-bit random difference pattern at the output. Thus, the complexity is close to an exhaustive search of the remaining candidates from random faults on the first and last row before the last two substitution layers. Then, when $n$ random faults (one chance out of two to flip each bit) have been injected on one row, the probability to obtain no difference on a nibble is equal to $1/2^n$ and the probability to obtain one difference is equal to $\sum_{i=1}^{n} 1/2^i = 1 - 1/2^n = (2^n - 1)/2^n$. Moreover, if we get no difference with all faults (on the first and last row) then, we still have 16 candidates for the corresponding nibble. On the other hand, if we get only one difference, we obtain 4 candidates. Finally, if we get the two differences, we retrieve the correct value.

Finally, the average number of remaining candidates for $k_0$ (resp. $k_1$ once $k_0$ have been recovered) from random faults before the last (resp. penultimate) linear layer, $n_1$ on the first row and $n_2$ on the last row, is equals to:

$$\left(\frac{16}{2^{n_1+n_2}} + \sum_{i=1}^{n_1} \frac{4}{2^{i+n_2}} + \sum_{i=1}^{n_2} \frac{4}{2^{i+n_1}} + \left(\sum_{i=1}^{n_1} \frac{1}{2^i}\right)\left(\sum_{i=1}^{n_2} \frac{1}{2^i}\right)\right)^{16}$$

or equivalently

$$\left(\frac{16 + 4(2^{n_1} - 1) + 4(2^{n_2} - 1) + (2^{n_1} - 1)(2^{n_2} - 1)}{2^{n_1+n_2}}\right)^{16}$$

or similarly

$$\left(\frac{9}{2^{n_1+n_2}} + \frac{3}{2^{n_1}} + \frac{3}{2^{n_2}} + 1\right)^{16}$$

As we can see, $n_1$ and $n_2$ are interchangeable. Moreover, for a given $n = n_1 + n_2$, the minimum of the previous equation is reached for $n_1 = \lfloor (n/2) \rfloor$ and $n_2 = \lceil (n/2) \rceil$. Table 1 shows the average number of remaining candidates for a subkey according to $n$ from $n_1 = \lfloor (n/2) \rfloor$ (resp. $n_2 = \lceil (n/2)) \rceil$ random faults on the first (resp. last) row of the linear layer input in the previous round.

Table 1: Remaining candidates $R$ for a subkey from $n$ random faults

| $n$ | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|
| $R$ | $2^{42.3}$ | $2^{25.8}$ | $2^{14.7}$ | $2^{7.9}$ | 17.6 | 4.33 | 2.10 | 1.45 | 1.21 | 1.10 |

*Note : we can also reduce the number of remaining candidates for $k_0$ from the faults obtained before the penultimate substitution layer as we have seen in the previous section.*

## 8 Countermeasures

In this section, we present and briefly analyze three possible countermeasures to thwart such attacks. The first one protects against correlation electromagnetic analysis, the second one against differential faults analysis and the last one against both but requires protocol modifications. This list of countermeasures is not exhaustive and any combination of those three can be used in practice.

### 8.1 Against correlation electromagnetic analysis

There are many strategies to protect a cipher from side channel attacks. At the software level, the most common countermeasure is masking, which consists in applying secret sharing at the implementation

level. Most of the proposed solutions are polynomial-based masking schemes in which multiplications over a binary finite field are secured using the ISW scheme [20]. In order to reduce the overhead introduced by this kind of countermeasure, bitslice masking has been recently proposed [18,16,31]. As the PRIDE S-box is designed for bitsliced implementation, we have naturally investigated this method. For a nibble denoted $n = a \,||\, b \,||\, c \,||\, d$, a mask of first order $m = m_a \,||\, m_b \,||\, m_c \,||\, m_d$ and $\widetilde{n} = n \oplus m = \widetilde{a} \,||\, \widetilde{b} \,||\, \widetilde{c} \,||\, \widetilde{d}$, the S-Box returns the output nibble $\widetilde{N} = \widetilde{A} \,||\, \widetilde{B} \,||\, \widetilde{C} \,||\, \widetilde{D}$ where

$$\begin{aligned}
\widetilde{A} &= \widetilde{c} \oplus (\widetilde{a} \cdot \widetilde{b}) \\
\widetilde{B} &= \widetilde{d} \oplus (\widetilde{b} \cdot \widetilde{c}) \\
\widetilde{C} &= \widetilde{a} \oplus (\widetilde{A} \cdot \widetilde{B}) \\
\widetilde{D} &= \widetilde{b} \oplus (\widetilde{B} \cdot \widetilde{C})
\end{aligned}$$

The challenging part of gate-level masking is to provide a construction for AND gates. Such a construction is proposed in [37]. It consists in introducing a random $r$ as a new mask and modifying the AND gate computation. For example, to compute $\widetilde{z} = \widetilde{a} \cdot \widetilde{b} = (a \oplus m_a) \cdot (b \oplus m_b)$ we will generate a random bit $r$ and compute:

$$m_z = r \tag{7}$$

$$\widetilde{z} = (\widetilde{a} \cdot \widetilde{b}) \oplus (m_a \cdot m_b) \oplus (m_a \cdot \widetilde{b}) \oplus (m_b \cdot \widetilde{a}) \oplus r \tag{8}$$

In the particular case of PRIDE, by using the method described above, we will need to generate 4 random bits $(r_A, r_B, r_C, r_D)$ for each secure AND gate to compute the updated mask $M = M_A \,||\, M_B \,||\, M_C \,||\, M_D$ where

$$\begin{aligned}
M_A &= m_c \oplus r_A \\
M_B &= m_d \oplus r_B \\
M_C &= m_a \oplus r_C \\
M_D &= m_b \oplus r_D
\end{aligned}$$

Concerning the $\mathcal{L}-$layer, as it is a linear operation, we just have to compute it over the state mask $M$ in parallel in order to be able to correctly unmask the masked state (*i.e.* to recover $N$ from $\widetilde{N}$ and $M$).

### 8.2 Against differential faults analysis

Making two computations for the last rounds is a simple countermeasure against this kind of attack. We save the state of the cipher $X_{18}$ in memory, possibly $k$ times for more security - as it concerns lightweight cryptography it 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 ($X_{18}$) and compare it 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. Otherwise, the execution performs normally. We can also apply a majority vote by duplicating the computations twice, possibly $2k$ times and give as output the one that appears most. Figure 9 shows a majority vote using duplication.

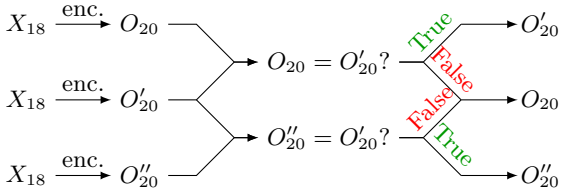


Figure 9: Majority vote using duplication

This countermeasure uses, for encryption and decryption, two additional matrix layers and three additional substitution layers, subkey updates and subkey additions per duplication. It introduces an overhead of 15% of the total PRIDE cost per duplication.

### 8.3 Against both

Another countermeasure proposed by Guilley *and al.* in [19] is to add a random mask to the message in order 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, which is XORed it with the asked plaintext and the corresponding ciphertext 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 an initialization made public. The LFSR thus generates $2^{64} - 1$ different masks. It must not be accessible to the user to avoid its reset. For that, 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 from getting two encryptions of the same plaintext, and therefore to run 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 one 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 device is only used for encryption and that decryption is carried out on a protected server. The second option 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.

In both cases, with same plaintext and key as inputs, the countermeasure protects against correlation power analysis (as the operations are not the same between two computations) and against differential faults analysis (as it does not return twice the same output). These two options are notexpensive but request a procedure constraint. Figure 10 illustrates the countermeasure.

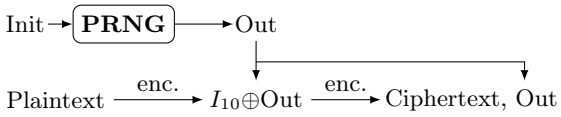


Figure 10: Mask based on the Guilley countermeasure

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

## 9 Conclusion

In this paper, we underline the importance of considering physical attacks when implementing lightweight cryptography by illustrating how passive and active physical attacks can be carried against a PRIDE software implementation. The results show that PRIDE is vulnerable to CEMA as well as DFA and so additional countermeasures are required when put into practice. Finally, we propose such countermeasures for both attacks. The next steps shall now be to analyse the countermeasures' effects in terms of security and performance.

## References


1. 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)
2. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side—Channel(s), pp. 29–45. Springer Berlin Heidelberg, Berlin, Heidelberg (2003), http://dx.doi.org/10.1007/3-540-36400-5_4
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. 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)
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. 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)
7. 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/
8. 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)
9. 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)
10. 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)
11. 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)
12. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model, pp. 16–29. Springer Berlin Heidelberg, Berlin, Heidelberg (2004), http://dx.doi.org/10.1007/978-3-540-28632-5_2
13. 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)
14. 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)
15. Gong, Z., Nikova, S., Law, Y.W.: RFID. Security and Privacy: 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers, chap. KLEIN: A New Family of Lightweight Block Ciphers, pp. 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-25286-0_1
16. Goudarzi, D., Rivain, M.: How fast can higher-order masking be in software? Cryptology ePrint Archive, Report 2016/264 (2016), http://eprint.iacr.org/2016/264
17. GÃľrard, B., Grosso, V., Naya-Plasencia, M., Standaert, F.X.: 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
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. 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

20. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks, pp. 463–481. Springer Berlin Heidelberg, Berlin, Heidelberg (2003), http://dx.doi.org/10.1007/978-3-540-45146-4_27
21. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. pp. 388–397. CRYPTO '99, Springer-Verlag, London, UK, UK (1999), http://dl.acm.org/citation.cfm?id=646764.703989
22. 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)
23. Lac, B., Beunardeau, M., Canteaut, A., Fournier, J.J.A., 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)
24. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
25. 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)
26. Piret, G., Roche, T., Carlet, C.: PICARO – A Block Cipher Allowing Efficient Higher-Order Side-Channel Resistance, pp. 311–328. Springer Berlin Heidelberg, Berlin, Heidelberg (2012), http://dx.doi.org/10.1007/978-3-642-31284-7_19
27. Poschmann, A., Moradi, A., Khoo, K., Lim, C.W., Wang, H., Ling, S.: Side-channel resistant crypto for less than 2, 300 ge. J. Cryptology 24, 322–345 (2011)
28. Quisquater, J., Samyde, D.: Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In: E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Berlin, Germany, Cannes, France (September 19-21, 2001)
29. Rebeiro, C., Mukhopadhyay, D.: Cryptanalysis of CLEFIA Using Differential Methods with Cache Trace Patterns, pp. 89–103. Springer Berlin Heidelberg, Berlin, Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-19074-2_7
30. Renauld, M., Standaert, F.X.: Combining Algebraic and Side-Channel Cryptanalysis against Block Ciphers. In: 30-th Symposium on Information Theory in the Benelux (5 2009)
31. Rivain, D.G.M.: On the multiplicative complexity of boolean functions and bitsliced higher-order masking. Cryptology ePrint Archive, Report 2016/557 (2016), http://eprint.iacr.org/2016/557
32. Selvam, R., Shanmugam, D., Annadurai, S.: Side channel attacks: Vulnerability analysis of prince and rectangle using dpa
33. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract), pp. 181–195. Springer Berlin Heidelberg, Berlin, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74619-5_12
34. Skorobogatov, S.: Semi-invasive attacks - A new approach to hardware security analysis. Technical Report 630, University of Cambridge (April 2005)
35. 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)
36. 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
37. Trichina, E.: Combinational logic design for aes subbyte transformation on masked data. Tech. rep., IACR report (2003)
38. 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
39. Yang, L., Wang, M., Qiao, S.: Side Channel Cube Attack on PRESENT, pp. 379–391. Springer Berlin Heidelberg, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-10433-6_25
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 Exploitable obtained faults

Table 2 (resp. Table 3) shows the faults we obtained from the electromagnetic injection between the last two (resp. the penultimate) substitution layers. For each fault, Table 2 (resp. Table 3) provides the

value of $\Delta Out_{20}$ (resp. $\Delta Out_{19}$), obtained from the correct and the faulty ciphertexts, which allowed us to retrieve the exact value of the fault and the value of $\Delta In_{20}$ (resp. $\Delta In_{19}$). Indeed, as the fault was injected in only one row, the positions and the values of the active nibbles in $\Delta Out_{20}$ (resp. $\Delta Out_{19}$) allowed us to derive the value of $\Delta In_{20}$ (resp. $\Delta In_{19}$) and then the value of the fault. Finally, some faults have corrupted two 8-bit instructions but remain exploitable as the fault model is on 16 bits.

Table 2: Faults obtained between the last two substitution layers

| Faulty ciphertext | Fault value | Fault position | Value of $\Delta Out_{20}$ | Value of $\Delta In_{20}$ |
|---|---|---|---|---|
| 0x83735baa7632ac9e | 0x0080 | 1-st row of $Y_{19}$ | 0xa000800000002000 | 0x8000800000008000 |
| 0x03f3d30276128c9e | 0xa8a8 | 2-nd row of $X_{20}$ | 0x6010c000c0606000 | 0x4040400040404000 |
| 0xcb339beaf67aacde | 0x0400 | 3-rd row of $Y_{19}$ | 0xcc0000000f000000 | 0x2200000002000000 |
| 0xc47397aaf23aa09e | 0xcf00 | 3-rd row of $X_{20}$ | 0xcc00df8800000000 | 0x2200222200000000 |
| 0xcb329beaf67aacde | 0x0081 | 3-rd row of $Y_{19}$ | 0xcc0000000f000008 | 0x2200000002000002 |
| 0xadd5df8ad21c88b8 | 0xa6a6 | 3-rd row of $X_{20}$ | 0xc0b00f8080f00bb0 | 0x2020022020200220 |
| 0x0b739f2276b22c96 | 0x004c | 4-th row of $Y_{19}$ | 0x7400040060007000 | 0x1100010010001000 |
| 0x0b730e41f793bcb4 | 0xbe00 | 4-th row of $Y_{19}$ | 0x0405040664707056 | 0x0101010111101011 |
| 0x0b73d3a276322496 | 0x8000 | 4-th row of $Y_{19}$ | 0x7000500000007000 | 0x1000100000001000 |
| 0x0b73c23377b33486 | 0x9999 | 4-th row of $X_{20}$ | 0x7005500660057006 | 0x1001100110011001 |
| 0x0b73a40f759b34be | 0x5a00 | 4-th row of $Y_{19}$ | 0x7445546660700406 | 0x1111111110100101 |
| 0x0b737b88f61aacbc | 0x0002 | 4-th row of $Y_{19}$ | 0x0040000000700050 | 0x0010000000100010 |
| 0x0b73eb1176933ca4 | 0x000b | 4-th row of $Y_{19}$ | 0x7045000060757056 | 0x1011000010111011 |

Table 3: Faults obtained between the penultimate two substitution layers

| Faulty ciphertext | Fault value | Fault position | Value of $\Delta Out_{19}$ | Value of $\Delta In_{19}$ |
|---|---|---|---|---|
| 0xb3035fae64aabc8e | 0x006a | 1-st row of $X_{19}$ | 0x0000000003208080 | 0x0000000008808080 |
| 0x3f6713aecea2948e | 0xc100 | 1-st row of $X_{19}$ | 0x8300000200000000 | 0x8800000800000000 |
| 0x1bdad38aff8aa4ae | 0x0039 | 1-st row of $X_{19}$ | 0x000000000022800a | 0x0000000000888008 |
| 0x3165d7eea5f5f4dc | 0x7f00 | 1-st row of $X_{19}$ | 0x03a88a8200000000 | 0x0888888800000000 |
| 0x16fdd78aea9ca890 | 0x000b | 2-nd row of $X_{19}$ | 0x000000000000a066 | 0x0000000000004044 |
| 0x077fdeba72a7d9da | 0xd100 | 2-nd row of $X_{19}$ | 0xa60c000100000000 | 0x4404000400000000 |
| 0x12f193ceee10a898 | 0x0087 | 2-nd row of $X_{19}$ | 0x00000000c0000166 | 0x0000000040000444 |
| 0x92f9c2927701dcdc | 0xd10b | 2-nd row of $X_{19}$ | 0xa60c00010000a066 | 0x4404000400004044 |
| 0x81791f6e017bd89e | 0x003c | 3-rd row of $X_{19}$ | 0x000000000088eb00 | 0x0000000000222200 |
| 0x827873a04d02ac8c | 0x0083 | 3-rd row of $X_{19}$ | 0x00000000800000fc | 0x0000000020000022 |
| 0xb05e37e04c63acec | 0x00d7 | 3-rd row of $X_{19}$ | 0x000000008b080bfc | 0x0000000022020222 |
| 0x411737ca9638aeba | 0x0600 | 3-rd row of $X_{19}$ | 0x00000dd000000000 | 0x0000022000000000 |
| 0x08bf2c2551e6f6bf | 0x7a00 | 3-rd row of $X_{19}$ | 0x0bedf0d000000000 | 0x0222202000000000 |
| 0x303fbc2c4076debe | 0xe200 | 3-rd row of $X_{19}$ | 0xebe000d000000000 | 0x2220002000000000 |
| 0xd4bfe13bb63fa8e8 | 0x00cb | 4-th row of $X_{19}$ | 0x0000000064006077 | 0x0000000011001011 |
| 0x91f0e1b0f632ada9 | 0x0063 | 4-th row of $X_{19}$ | 0x0000000004400077 | 0x0000000001100011 |
| 0xc42ec0dbb65e18db | 0x00fd | 4-th row of $X_{19}$ | 0x0000000064446407 | 0x0000000011111101 |
| 0x4cbfd8ca365e88d2 | 0x00fc | 4-th row of $X_{19}$ | 0x0000000064446400 | 0x0000000011111100 |
| 0x856cc59ff218d813 | 0x004d | 4-th row of $X_{19}$ | 0x0000000004006407 | 0x0000000001001101 |

Now we present the faults that give as much information as all other. Table 4 shows all sets of candidates obtained for each nibble $\text{Nib}_i$ of $k_0 \oplus \mathcal{P}^{-1}(C)$ with $i \in \{0, \cdots, 15\}$, from faults injected between the last two substitution layers. Symbol $\emptyset$ means that the fault did not provide any information about the nibble (*i.e.* the 16 values are possible). Then, Table 5 shows all sets of candidates obtained for each nibble $\text{Nib}_i$ of $\left(\mathcal{P}^{-1} \circ \mathcal{L}-\mathsf{layer}^{-1}\right)\left(\mathcal{S}-\mathsf{layer}\big(C \oplus \mathcal{P}(k_0)\big) \oplus f_{20}(k_1)\right)$ with $i \in \{0, \cdots, 15\}$, from faults injected between the penultimate two substitution layers. We again denote by $\emptyset$ cases where the fault did not provide any information about the nibble (*i.e.* the 16 values are possible).

Table 4: Sets of candidates obtained from faults injected between the last two substitution layers

| Value of ($\Delta O_{20}$, $\Delta I_{20}$) | $\text{Nib}_0$ | $\text{Nib}_1$ | $\text{Nib}_2$ | $\text{Nib}_3$ | $\text{Nib}_4$ | $\text{Nib}_5$ | $\text{Nib}_6$ | $\text{Nib}_7$ | $\text{Nib}_8$ | $\text{Nib}_9$ | $\text{Nib}_{10}$ | $\text{Nib}_{11}$ | $\text{Nib}_{12}$ | $\text{Nib}_{13}$ | $\text{Nib}_{14}$ | $\text{Nib}_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (0xa000800000002000, 0x8000800000008000) | 0x1 0x3 0x9 0xb | ∅ | ∅ | ∅ | 0x5 0x6 0xd 0xe | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | 0x0 0x2 0x8 0xa | ∅ | ∅ | ∅ |
| (0xcc00df8800000000, 0x2200222200000000) | 0x5 0x9 | 0x5 0x9 | ∅ | ∅ | 0x6 0xb | 0x1 0xe | 0x0 0x2 0x8 0xa | 0x0 0x2 0x8 0xa | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| (0xcc0000000f000008, 0x2200000002000002) | 0x5 0x9 | 0x5 0x9 | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | 0x1 0xe | ∅ | ∅ | ∅ | ∅ | ∅ | 0x0 0x2 0x8 0xa |
| (0xc0b00f8080f00bb0, 0x2020022020200220) | 0x5 0x9 | ∅ | 0x4 0x7 0xc 0xf | ∅ | ∅ | 0x1 0xe | 0x0 0x2 0x8 0xa | ∅ | 0x0 0x2 0x8 0xa | ∅ | 0x1 0xe | ∅ | ∅ | 0x4 0x7 0xc 0xf | 0x4 0x7 0xc 0xf | ∅ |
| (0x0405040664707056, 0x0101010111101011) | ∅ | 0x0 0x1 0x4 0x5 | ∅ | 0x2 0x3 0x6 0x7 | ∅ | 0x0 0x1 0x4 0x5 | ∅ | 0xa 0xb 0xc 0xd | 0xa 0xb 0xc 0xd | 0x0 0x1 0x4 0x5 | 0x8 0x9 0xe 0xf | ∅ | 0x8 0x9 0xe 0xf | ∅ | 0x2 0x3 0x6 0x7 | 0xa 0xb 0xc 0xd |
| (0x7005500660057006, 0x1001100110011001) | 0x8 0x9 0xe 0xf | ∅ | ∅ | 0x2 0x3 0x6 0x7 | 0x2 0x3 0x6 0x7 | ∅ | ∅ | 0xa 0xb 0xc 0xd | 0xa 0xb 0xc 0xd | ∅ | ∅ | 0x2 0x3 0x6 0x7 | 0x8 0x9 0xe 0xf | ∅ | ∅ | 0xa 0xb 0xc 0xd |
| (0x7445546660700406, 0x1111111110100101) | 0x8 0x9 0xe 0xf | 0x0 0x1 0x4 0x5 | 0x0 0x1 0x4 0x5 | 0x2 0x3 0x6 0x7 | 0x2 0x3 0x6 0x7 | 0x0 0x1 0x4 0x5 | 0xa 0xb 0xc 0xd | 0xa 0xb 0xc 0xd | 0xa 0xb 0xc 0xd | ∅ | 0x8 0x9 0xe 0xf | ∅ | ∅ | 0x0 0x1 0x4 0x5 | ∅ | 0xa 0xb 0xc 0xd |

Table 5: Sets of candidates obtained from faults injected between the penultimate two substitution layers

| Value of ($\Delta Y_{19}$, $\Delta X_{19}$) | $\text{Nib}_0$ | $\text{Nib}_1$ | $\text{Nib}_2$ | $\text{Nib}_3$ | $\text{Nib}_4$ | $\text{Nib}_5$ | $\text{Nib}_6$ | $\text{Nib}_7$ | $\text{Nib}_8$ | $\text{Nib}_9$ | $\text{Nib}_{10}$ | $\text{Nib}_{11}$ | $\text{Nib}_{12}$ | $\text{Nib}_{13}$ | $\text{Nib}_{14}$ | $\text{Nib}_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (0x03a88a8200000000, 0x0888888800000000) | ∅ | 0x4 0x7 0xc 0xf | 0x1 0x3 0x9 0xb | 0x5 0x6 0xd 0xe | 0x5 0x6 0xd 0xe | 0x1 0x3 0x9 0xb | 0x5 0x6 0xd 0xe | 0x0 0x2 0x8 0xa | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| (0x8300000200000000, 0x8800000800000000) | 0x5 0x6 0xd 0xe | 0x4 0x7 0xc 0xf | ∅ | ∅ | ∅ | ∅ | ∅ | 0x0 0x2 0x8 0xa | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| (0x0000000003208080, 0x0000000008808080) | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | 0x4 0x7 0xc 0xf | 0x0 0x2 0x8 0xa | ∅ | 0x5 0x6 0xd 0xe | ∅ | 0x5 0x6 0xd 0xe | ∅ |
| (0xa60c00010000a066, 0x4404000400004044) | 0x7 0xd | 0x8 0x9 0xe 0xf | ∅ | 0x6 0xa | ∅ | ∅ | ∅ | 0x0 0x1 0x4 0x5 | ∅ | ∅ | ∅ | ∅ | 0x7 0xd | ∅ | 0x8 0x9 0xe 0xf | 0x8 0x9 0xe 0xf |
| (0x0bedf0d000000000, 0x0222202000000000) | ∅ | 0x4 0x7 0xc 0xf | 0x3 0xd | 0x6 0xb | 0x1 0xe | ∅ | 0x6 0xb | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| (0x000000008b080bfc, 0x0000000022020222) | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | 0x0 0x2 0x8 0xa | 0x4 0x7 0xc 0xf | ∅ | 0x0 0x2 0x8 0xa | ∅ | 0x4 0x7 0xc 0xf | 0x1 0xe | 0x5 0x9 |
| (0x00000dd000000000, 0x0000022000000000) | ∅ | ∅ | ∅ | ∅ | ∅ | 0x6 0xb | 0x6 0xb | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| (0x0000000064446407, 0x0000000011111101) | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | 0xa 0xb 0xc 0xd | 0x0 0x1 0x4 0x5 | 0x0 0x1 0x4 0x5 | 0x0 0x1 0x4 0x5 | 0xa 0xb 0xc 0xd | 0x0 0x1 0x4 0x5 | ∅ | 0x8 0x9 0xe 0xf |

## B Proof of Proposition 1

$$
\begin{aligned}
\mathsf{Corr}(X,\ Y_i) &= \frac{\mathsf{Cov}(X,\ Y_i)}{\sigma_X\ \sigma_{Y_i}} \\
&= \frac{\mathsf{E}(X\ Y_i) - \mathsf{E}(X)\ \mathsf{E}(Y_i)}{\sigma_X\ \sqrt{\mathsf{E}\Big(\big(Y_i - \mathsf{E}(Y_i)\big)^2\Big)}} \\
&= \frac{\mathsf{E}\big(X\ (z - Y_j)\big) - \mathsf{E}(X)\ \mathsf{E}(z - Y_j)}{\sigma_X\ \sqrt{\mathsf{E}\Big(\big(z - Y_j - \mathsf{E}(z - Y_j)\big)^2\Big)}} \\
&= \frac{-\mathsf{E}\big(X\ Y_j)\big) + \mathsf{E}(X)\ \mathsf{E}(Y_j)}{\sigma_X\ \sqrt{\mathsf{E}\Big(\big(-Y_j + \mathsf{E}(Y_j)\big)^2\Big)}} \\
&= \frac{-\big(\mathsf{E}\big(X\ Y_j)\big) - \mathsf{E}(X)\ \mathsf{E}(Y_j)\big)}{\sigma_X\ \sqrt{\mathsf{E}\Big(\big(Y_j - \mathsf{E}(Y_j)\big)^2\Big)}} \\
&= \frac{-\mathsf{Cov}(X,\ Y_i)}{\sigma_X\ \sigma_{Y_i}} \\
&= -\mathsf{Corr}(X,\ Y_j)
\end{aligned}
$$

## C S-box formulation

$$
\begin{aligned}
A &= c \oplus (a \,\&\, b) \\
B &= d \oplus (b \,\&\, c) \\
C &= a \oplus (A \,\&\, B) \\
D &= b \oplus (B \,\&\, C)
\end{aligned}
$$

## D C source code

### D.1 Key addition layer

```
void key_add_layer(unsigned char key[8], unsigned char state[8]){
        // key schedule
        key[1] +=  193;
        key[3] +=  165;
        key[5] +=  81;
        key[7] +=  197;
        // key addition
        state[0] ^= key[0];
        state[1] ^= key[1];
        state[2] ^= key[2];
        state[3] ^= key[3];
        state[4] ^= key[4];
        state[5] ^= key[5];
        state[6] ^= key[6];
        state[7] ^= key[7];
}
```

Listing 1.1: Key addition layer C source code

### D.2 $\mathcal{S}-$layer and $\mathcal{L}-$layer

```c
void s_layer(unsigned char state[8]){
        unsigned char tmp0, tmp1, tmp2, tmp3;
        // saves the input state
        tmp0 = state[0];
        tmp1 = state[1];
        tmp2 = state[2];
        tmp3 = state[3];
        // a & b
        state[0] &= state[2];
        state[1] &= state[3];
        // A = c ^(a & b)
        state[0] ^= state[4];
        state[1] ^= state[5];
        // b & c
        state[2] &= state[4];
        state[3] &= state[5];
        // b = d ^ (b & c)
        state[2] ^= state[6];
        state[3] ^= state[7];
        // c = A
        state[4] = state[0];
        state[5] = state[1];
        // d = B
        state[6] = state[2];
        state[7] = state[3];
        // A & B
        state[4] &= state[6];
        state[5] &= state[7];
        // C = a ^ (A & B)
        state[4] ^= tmp0;
        state[5] ^= tmp1;
        // B & C
        state[6] &= state[4];
        state[7] &= state[5];
        // D = b ^ (B & C)
        state[6] ^= tmp2;
        state[7] ^= tmp3;
    }
```

Listing 1.2: $\mathcal{S}-$layer C source code

```c
void l_layer(unsigned char state[8]){
        unsigned char tmp0, tmp1, tmp2;
        // applies L0 matrix
        tmp0        = state[0];
        tmp1        = state[1];
        tmp2        = state[0] << 4;
        tmp2        |= state[0] >> 4;
        state[0]    = tmp2;
        tmp2        = state[1] << 4;
        tmp2        |= state[1] >> 4;
        state[1]    = tmp2;
        state[0]    ^= state[1];
        tmp0        ^= state[0];
        state[1]    = tmp0;
        state[0]    ^= tmp1;
        // applies L1 matrix
        tmp0        = state[3] << 4;
        tmp0        |= state[3] >> 4;
        state[3]    = tmp0;
        tmp0        = state[2] << 1;
        tmp0        |= state[2] >> 7;
        tmp1        = state[3] >> 1;
        tmp1        |= state[3] << 7;
        state[2]    ^= tmp1;
        tmp1        = state[2];
        state[2]    ^= tmp0;
        state[3]    ^= tmp1;
        // applies L2 matrix
        tmp0        = state[4] << 4;
        tmp0        |= state[4] >> 4;
        state[4]    = tmp0;
        tmp0        = state[4] << 1;
        tmp0        |= state[4] >> 7;
        tmp1        = state[5] >> 1;
        tmp1        |= state[5] << 7;
        state[4]    ^= tmp1;
        tmp1        = state[4];
        state[4]    ^= tmp0;
        state[5]    ^= tmp1;
        // applies L3 matrix
        tmp0        = state[6];
        tmp1        = state[7];
        tmp2        = state[6] << 4;
        tmp2        |= state[6] >> 4;
        state[6]    = tmp2;
        tmp2        = state[7] << 4;
        tmp2        |= state[7] >> 4;
        state[7]    = tmp2;
        state[6]    ^= state[7];
        tmp1        ^= state[6];
        state[7]    = tmp1;
        state[6]    ^= tmp0;
    }
```

Listing 1.3: $\mathcal{L}-$layer C source code