There is Wisdom in Harnessing the Strengths of your Enemy: Customized Encoding to Thwart Side-Channel Attacks – Extended Version* –

Houssem Maghrebi, Victor Servant, Julien Bringer

SAFRAN Morpho,
18, Chaussée Jules César, 95520 Osny, France.
firstname.lastname@morpho.com

Abstract. Side-channel attacks are an important concern for the security of cryptographic algorithms. To counteract it, a recent line of research has investigated the use of software encoding functions such as dual-rail rather than the well known masking countermeasure. The core idea consists in encoding the sensitive data with a fixed Hamming weight value and perform all operations following this fashion. This new set of countermeasures applies to all devices that leak a function of the Hamming weight of the internal variables. However when the leakage model deviates from this idealized model, the claimed security guarantee vanishes. In this work, we introduce a framework that aims at building customized encoding functions according to the precise leakage model based on stochastic profiling. We specifically investigate how to take advantage of adversary's knowledge of the physical leakage to select the corresponding optimal encoding. Our solution has been evaluated within several security metrics, proving its efficiency against side-channel attacks in realistic scenarios. A concrete experimentation of our proposal to protect the PRESENT Sbox confirms its practicability. In a realistic scenario, our new custom encoding achieves a hundredfold reduction in leakage compared to the dual-rail, although using the same code length.

Keywords: constant weight countermeasures, stochastic characterization, customized encoding function, security metrics, information theoretic analysis, side-channel analysis.

1 Introduction

Side-Channel Attacks. Side-Channel attacks (SCA) are nowadays well known and most designers of secure embedded systems are aware of them. Since the first

* This paper is an extended version of a previous publication that will appear in the proceedings of the 23rd International Conference on Fast Software Encryption (FSE 2016). This version includes the Appendices that were elided for space.

public reporting of these threats [15], a lot of effort has been devoted towards the research about side-channel attacks and the development of corresponding countermeasures. Side-channel attacks exploit information leaked from the physical implementations of cryptographic algorithms. Since, this leakage (*e.g.* the power consumption or the electromagnetic emanations) depends on the internally used secret key, the adversary may perform an efficient key-recovery attack to reveal this sensitive data. As this property can be exploited with relatively cheap equipment, these attacks pose a serious practical threat to cryptographic embedded systems. Amongst the side-channel attacks, two classes may be distinguished:

- The set of so-called *profiling SCA*: is the most powerful kind of SCA attacks and consists of two steps. First, the adversary procures a copy of the *target device* and uses it to characterize the physical leakage. Second, he performs a key-recovery attack on the target device. This set of profiled attacks includes Template attacks [5] and Stochastic models (*a.k.a.* Linear Regression Analysis) [10,22,23].
- The set of so-called *non-profiling SCA*: corresponds to a much weaker adversary who has only access to the physical leakage captured on the target device. To recover the secret key used, he performs some statistical analyses to detect the dependency between the leakage measurements and this sensitive variable. This set of non-profiled attacks includes Differential Power Analysis (DPA) [15], Correlation Power Analysis (CPA) [3] and Mutual Information Analysis (MIA) [13].

Side-Channel Countermeasures. A deep look at the state-of-the-art shows that several countermeasures have been published to deal with side-channel attacks. Amongst SCA countermeasures, two classes may be distinguished [18]:

- The set of so-called *masking countermeasures*: the core principle of masking is to ensure that every sensitive variable is randomly split into at least two shares so that the knowledge of a strict sub-part of the shares does not give information on the shared variable itself. Masking can be characterized by the number of random masks used per sensitive variable. So, it is possible to give a general definition for a d^{th}-order masking scheme: every sensitive variable Z is randomly split into $d+1$ shares $M_0, \cdots, M_d$ in such a way that the relation $M_0 \perp \cdots \perp M_d = Z$ is satisfied for a group operation $\perp$ (*e.g.* the XOR operation used in the *Boolean masking*, denoted as $\oplus$) and no tuple of strictly less than $d+1$ shares depends on Z. In the literature, several provably secure higher-order masking schemes have been proposed, see for instance [9], [12] and [21].
- The set of so-called *hiding countermeasures*: the core idea consists in making the activity of the physical implementation constant by either adding complementary logic to the existing logic [7] (in a hardware setting) or by using a *specific encoding* of the sensitive data [6,14,24] (in a software setting). Therefore, making this activity constant would theoretically remove the correlation between the leakage measurements and the secret key.

Constant Weight Countermeasures. A recent line of works has investigated possibilities to protect block ciphers in software implementations using *constant weight coding* rather than using masking techniques. It is a specific encoding that has the particularity that all its codewords have a constant Hamming weight. More precisely, Hoogvorst *et al.* in [14] have presented a *dual-rail* implementation of PRESENT [2] in 2011. The idea was straightforwardly taken from existing dual-rail hardware, and consists in encoding one bit *s.t.* the logical value 0 is represented as 01 and the logical value 1 is represented as 10 (or the inverse).
Another idea derived from dual-rail can be found in a work by Chen *et al.* [6]. Several encodings are used, by reordering the bits and their complements in a word, in order to ensure constant Hamming weight and distance leakage for all operations of the block cipher PRINCE.

Recently, at CARDIS 2014, Servant *et al.* in [24] have proposed a new constant weight implementation of the AES extending the idea of the software dual-rail countermeasure proposed by Hoogvorst *et al.* in [14]. The core idea consists in encoding efficiently the sensitive data as a whole (*i.e.* not bit per bit) with a fixed Hamming weight value and then performing the AES internal operations following this fashion. When assuming a Hamming weight leakage model, the authors proved that their proposal is a leak-free countermeasure. However real world devices do not fit this model, as explained hereafter.

Stochastic Characterization of the Leakage. It is often assumed that a device leaks information based on the Hamming weight of the processed data. This assumption is quite realistic and many security analyses in the literature have been conducted following this model [3,19]. However, this assumption is not complete in real hardware [28], due to small load imbalances, process variations, routing, *etc.* For instance, authors in [16] have characterized, using a *stochastic approach*, the leakage of four AES Sbox outputs when implemented in three different devices. The obtained results prove that the leakage is very unbalanced for each Sbox output and hence, the Hamming weight assumption is unsound in practice. This imbalance always leaks some information that can be exploited by a SCA adversary. Hence, the security guarantee claimed by constant weight countermeasures does not necessarily hold in real world.

Our Contribution. In this paper, we refine the notion of data encoding as a countermeasure to thwart side-channel attacks. In fact, we try to bridge the gap between the physical leakage characteristics and the optimal encoding which balances at best the data leakage. This work exposes a method based on a first precise stochastic characterization of the target device, followed by the generation of a specific encoding according to this model. To do so, we propose an algorithm to select the best encoding function according to the physical leakage characterized on the target device. Our experiments show that the proposed encoding framework is more efficient than the existing constant weight countermeasures. We theoretically prove that our proposal reduces the Signal-to-Noise Ratio and hence, an adversary requires more traces to disclose the secret key than on the existing constant weight countermeasures. Furthermore, the security

evaluation conducted illustrates that the leaked information is minimal and the efficiency of stochastic attack in exploiting this leakage is reduced drastically. Finally, the practical assessment of our proposal confirms its practicability to protect cryptographic operations. When device registers leak roughly the same function, our proposal could be applied to fully protect a block cipher. This assumption is not fully realistic, meanwhile, our work is a first step towards protecting block ciphers by involving customized encoding. If registers happen to leak vastly differently, then we need different encodings for each register and code conversions between memory accesses to protect a whole block cipher.

Paper Outline. The paper is organized as follows. In Sec. 2, we first detail two published constant weight implementations to protect a block cipher in a software setting. Then, we describe our new encoding framework in Sec. 3 and provide a theoretical analysis of it in Sec. 4. Furthermore, an information theoretic analysis and a security evaluation are conducted in Sec. 5. This is followed by some practical attack experiments applied on real devices in Sec. 6. Finally, Sec. 7 draws general conclusions and opens some perspectives for future work.

2 Existing Works on Leakage Balancing by Involving Encoding Functions

This principle of data internal encodings has already been proposed by Chow *et al.* in [8] in the context of *white-box cryptography*. Since then, several countermeasures have been proposed aiming at balancing the leakage by using some constant Hamming weight encodings in a *grey-box context*[1]. For instance, Hoogvorst *et al.* [14] have adapted the hardware dual-rail countermeasure to protect a software implementation of PRESENT. To do so, the authors suggest to duplicate the bit values representation, *i.e.* to use two bits to represent the logical value of one bit. For instance, one can encode the logical value 0 as 01 and the logical value 1 as 10 (or the inverse). When applying such an encoding to protect a n-bit variable, all codewords generated have a constant Hamming weight of n. Hence, assuming a Hamming weight leakage model, the power consumption provides no sensitive information. For instance, the code used to protect a 4-bit variable is presented in Appendix A, where all codewords have a constant weight of 4. In the sequel, it will be referred as the dual-rail code.

The dual-rail representation is a specific case of this class of constant weight codes, but it is not the only option one should consider in a software setting. As a first example, authors of [6] propose a variation of the dual-rail applied to the block cipher PRINCE. Another example is [24], in which the authors propose a new balancing strategy based on the use of a code with the smallest cardinal available to encode the sensitive data. To protect a 4-bit variable, one can use 16 codewords of 6-bit length, each with a constant Hamming weight of 3. The used

[1] The adversary has access to the inputs and outputs of the cryptographic algorithm plus extra side-channel information.

code is provided in Appendix A. It will be referred as the (3,6)-code in the rest of this paper. The security analysis conducted in [24] proves that this constant weight implementation is a leak-free countermeasure under a Hamming weight leakage model assumption. However, when the leakage function deviates from this idealized model, the security guarantee provided by this countermeasure vanish as discussed in this reference.

To sum up, all these investigations on how to balance the physical leakage were conducted under the Hamming weight leakage model and with no prior characterization of the target device to incorporate the precise leakage model. Moreover, the choice of the code is made independently of the real bit leakage (for example, in dual-rail representation, the logical value 1 is usually encoded as 10). Therefore, the claimed security level of these countermeasures could not be obtained in practice, where the bits may leak differently [16].

In the following section, we propose a framework for protecting sensitive data by using specific encoding. It is aimed to bridge the physical leakage characteristics to the choice of an optimal encoding function.

3 Towards a New Encoding Procedure for Leakage Balancing

Unlike previous works in which the Hamming weight model is often assumed, our solution is essentially based on harnessing the leakage characteristics and building a customized encoding accordingly to obtain the best balanced leakage. So, our framework is composed of two steps detailed in the following subsections.

3.1 First Step: Stochastic Characterization of the Leakage Function

A primordial step in our proposed framework is to take advantage of the adversary's knowledge of the target device during a *stochastic characterization phase*, *a.k.a. leakage profiling*.

Let Z be a sensitive variable defined over $\mathbb{F}_2^n$, then a stochastic characterization assumes that the leakage function $\mathcal{L}(Z)$ can be expressed as the sum of two *mutually independent* parts:

- a deterministic part $\mathcal{D}(Z)$: a function representing the power consumption during the processing of the sensitive variable Z and,
- a random part $\mathcal{R}$: a Gaussian noise with null mean and standard deviation σ.

Hence, the leakage function can be rewritten: $\mathcal{L}(Z) = \mathcal{D}(Z) + \mathcal{R} = \sum_{i=1}^{u} \alpha_i \delta_i(Z) + \mathcal{N}(0, \sigma)$, where α_i are some weighting coefficients and δ_i are some well chosen *basis functions*. Besides, we stress the fact that the basis choice is essential since it directly impacts the profiling efficiency.

For the sake of simplicity, in this work, we assume a linear basis. This choice is also motivated by the fact that higher-order basis functions are playing a minor role despite their better representation of the reality [10]. Moreover, the

deterministic part of the leakage in practice is very close to the value of the linear part as discussed in [10]. So, our goal here is to characterize the leakage function when its deterministic part deviates from the Hamming weight model, but keeps the same degree. The study of higher-order basis functions (*e.g.* quadratic, cubic, ...) is out of the scope of this paper.

This implies that every bit of the sensitive variable leaks independently. This assumption is often used in SCA context to characterize the perceived device leakage and sometimes referred as *Independent Bit Leakage* (IBL) assumption [10]. We recall hereafter this assumption.

Assumption 1 ***(IBL assumption)*** *Let Z be a sensitive variable defined over $\mathbb{F}_2^n$, then the deterministic part of the leakage function can be rewritten: $\mathcal{D}(Z) = \sum_{i=1}^{n} \alpha_i Z[i]$, where $Z[i]$ denotes the i^{th} bit of the sensitive variable Z.*

Under Assumption 1, the leakage function can be rewritten:

$$\mathcal{L}(Z) = \sum_{i=1}^{n} \alpha_i Z[i] + \mathcal{N}(0, \sigma) \ . \tag{1}$$

So to recover the leakage function, one can apply a *linear regression* [10,16] to obtain a precise estimation of the α_i coefficients under the IBL assumption.

3.2 Second Step: Encoding Function Selection

Once the leakage function is characterized, the second step of our framework consists in applying Algorithm 1 to obtain the optimal encoding function *w.r.t.* the profiled leakage.

Algorithm 1 Selection of the optimal encoding function

Input: m: the codeword bit-length, n: the sensitive variable bit-length and α_i: the leakage bit weights, where i in $[\![1, m]\!]$
Output: 2^n codewords of m-bit length

1: **for** X in $[\![0, 2^m - 1]\!]$ **do**
2: Compute the power consumption for each codeword X and store the result in table D: $D[X] = \sum_{i=1}^{m} \alpha_i X[i]$
3: Store the corresponding value of the codeword in the index table I: $I[X] = X$
4: **end for**
5: Sort the power consumption stored in table D and the index table I accordingly
6: **for** j in $[\![0, 2^m - 2^n]\!]$ **do**
7: Find the $argmin$ of $|D[j] - D[j + 2^n]|$
8: **end for**
9: **return** 2^n codewords corresponding to $[\![I[argmin], I[argmin + 2^n]]\!]$

Our Algorithm 1 takes as inputs: the length in bits of respectively the codewords and the sensitive data and, for each bit, the corresponding leakage weight

obtained during a stochastic profiling as explained in the previous subsection. Then, it outputs 2^n codewords such that the delta consumption is the lowest among all subsets of 2^n codewords. Since the bit weights are unbalanced in practice, we argue that finding a code that guarantees a perfectly constant leakage remains an unreachable goal in most of cases.

Given the output codewords length, we compute the expected power consumption for each codeword and we store the result and the codeword value in table D and table I respectively (*c.f.* the first loop from Line 1 to Line 4 in Algorithm 1). Then, we sort table D (in ascending or descending order) and the index table I accordingly (*c.f.* Line 5 in Algorithm 1). Finally, since our goal is to choose a subset of 2^n codewords such that the delta consumption is the lowest one, we compute the delta of consumption for each subset of 2^n elements (*c.f.* the last loop from Line 6 to Line 8 in Algorithm 1) and later we select 2^n indexes from table I that minimize this delta. Thus, we obtain a code that ensures the best balancing of the leakage *w.r.t.* the stochastic profiling result.

A clustering algorithm [1] would also give good results for this problem, but we explain hereafter why we chose this algorithm which is somewhat simpler to analyze. Let d be the maximum distance between two elements of a set S of n elements. One can show that $Var(S) < n.d^2$, so that intuitively, minimizing this distance d gives a subset with one of the lowest variances (and hence, one of the lowest SNR). There might be a set S' with lower variance but higher distance d', but in that case it would be easier to distinguish the two extreme values of this set. Some attacks might use this fact to improve the success rate.

Our framework consequently helps building properly encoding function customized for the physically observable leakage. It acts as an interface between the adversary's knowledge of the physical leakage and the optimal encoding to be used accordingly. We stress the fact that our Algorithm 1 is still applicable if the IBL assumption is not respected. To do so, one should inject the obtained leakage function in Line 2 and execute the algorithm to carry out the encoding function.

4 Theoretical Analysis of the New Customized Encoding

In what follows, we provide a theoretical analysis of our solution. Namely, we will show that to succeed a *first-order univariate correlation attack* on our proposal, an adversary requires much more traces than on the existing constant weight countermeasures. This is due to the fact that the selected subset of codewords has a close-to-lowest power consumption variance among all possible subsets.

Let us start our analysis by exhibiting the explicit relationship between two *security metrics*: the Minimum number of Traces to Disclose the key with a given percentage of success rate (MTD), and the Signal-to-Noise Ratio (SNR). This link has already been demonstrated by Mangard in [17] for unprotected implementation. Our purpose is to provide the link between these two security metrics for encoding-based countermeasures.

To do so, we first recall how the number of traces to disclose the key is connected to the Correlation Power Analysis (CPA).

4.1 Analytical Derivation of the Security Level for Correlation Attacks.

The CPA attack [3] is based on the computation of the *Pearson correlation coefficient* between the leakage function $\mathcal{L}(Z)$ and a *prediction function* $f(Z)$ chosen according to some assumptions on the device leakage model (*e.g.* the Hamming weight function). Hence, the Pearson correlation coefficient can be rewritten:

$$\rho[\mathcal{L}(Z); f(Z)] = \frac{\mathsf{Cov}[\mathcal{L}(Z); f(Z)]}{\sigma_{\mathcal{L}(Z)}\sigma_{f(Z)}} \ ,$$

where $\mathsf{Cov}[.;.]$ is the covariance and $\sigma_{\mathcal{L}(Z)}$ and $\sigma_{f(Z)}$ are respectively the standard deviation of the physical leakage and the prediction leakage function. Besides, in [17] the author demonstrated that the number of curves required to break a cryptographic implementation by CPA is equal to:

$$N_{1-\beta} = 3 + 8\left(\frac{Z_{1-\beta}}{\ln\left(\frac{1+\rho}{1-\rho}\right)}\right)^2 \ , \tag{2}$$

where $Z_{1-\beta}$ is a quantile of a normal distribution for the 2-sided confidence interval with error $1-\beta$. [2]

We introduce hereafter the *optimal correlation function* and exhibit its relationship to the SNR security metric. Then, we deduce the explicit link between the number of traces to disclose the key and the SNR.

4.2 From Optimal Correlation Function to the SNR

The optimal correlation function is defined as the function that maximizes the correlation $\rho[\mathcal{L}(Z); f(Z)]$ and can be obtained from Corollary 8 in [20]:

$$\rho_{\text{opt}} = \sqrt{\frac{\mathsf{Var}[\mathsf{E}[\mathcal{L}(Z) \mid Z = z]]}{\mathsf{Var}[\mathcal{L}(Z)]}} \ , \tag{3}$$

where $\mathsf{E}[.]$ and $\mathsf{Var}[.]$ denote the mean and the variance function respectively. Based on this definition, we introduce the following proposition.

Proposition 1. *Let $\mathcal{L}(Z)$ satisfies Eq.* (1)*. Then, the optimal correlation function satisfies:*

$$\rho_{opt} = \sqrt{\frac{1}{1+\frac{1}{SNR}}} \ , \tag{4}$$

[2] Some values of quantiles are given in Appendix B.

where the SNR can be rewritten:

$$SNR = \left(\mathsf{Var}[\sum_{i=1}^{n} \alpha_i Z[i]] \right) / \sigma^2 \ . \tag{5}$$

Proof. The proof of Proposition 3 is given in Appendix. C. □

As a direct consequence of Proposition 3, one can inject Eq. (C.1) into Eq. (2) to find the number of traces required by a CPA attack to succeed according to the SNR. Thus, assuming ρ is small[3], it yields the number of traces to achieve a success rate of 90%, denoted $N_{90\%}$:

$$N_{90\%} \approx 8 \left(\frac{Z_{90\%}}{2\rho} \right)^2 \approx 8 \left(\frac{Z_{90\%}}{2\sqrt{\frac{1}{1+\frac{1}{SNR}}}} \right)^2 \approx \frac{2Z_{90\%}^2}{SNR} \tag{6}$$

From Eq. (6) one can conclude that the smaller the SNR is, the more traces are required to achieve a success rate of 90% for a CPA attack. As a direct consequence, if we decrease the SNR by a factor X, then the required number of traces to succeed the CPA attack will be multiplied by X.

In the next subsection, we evaluate our proposal by computing the SNR and then deducing the $N_{90\%}$.

4.3 Evaluation of our Proposal within the SNR and the $N_{90\%}$ Security Metrics

We recall that the deterministic part of the leakage function, defined under the IBL assumption, satisfies $\mathcal{D}(Z) = \sum_{i=1}^{n} \alpha_i Z[i]$. In the sequel, we make an additional assumption on the statistical distribution of the bit leakage weights α_i. In fact, for the sake of simplicity, the distribution of the α_i coefficients can fairly be approximated by a Gaussian law. This assumption that we shall call Gaussian Bit Leakage Weight (GBLW) assumption is formalized hereafter.

Assumption 2 ***(GBLW assumption)*** *The bit leakage weights α_i are mutually independent random variables drawn from a Gaussian distribution with unity mean and standard deviation σ_α.*

Under Assumption 2, the leakage function can be rewritten:

$$\mathcal{L}(Z) = \boldsymbol{\alpha} \cdot Z + \mathcal{N}(0, \sigma) \ , \tag{7}$$

where $(\cdot)$ denotes the scalar product operation and $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]$ denotes the bit leakage weight vector such that for every i in $[\![1, n]\!]$ we have $\alpha_i \sim \mathcal{N}(1, \sigma_\alpha)$.

[3] In fact, we can approximate $\ln\left(\frac{1+\rho}{1-\rho}\right) = \ln(1+\rho) - \ln(1-\rho) \approx \rho - (-\rho) \approx 2\rho$.

Let $\mathcal{C}$ be a (n, m)-function, *i.e.* $\mathcal{C} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ *s.t.* $n \leq m$, denoting the encoding operation used to protect a sensitive variable Z in $\mathbb{F}_2^n$. Then, the leakage function can be expressed as:

$$\mathcal{L}(Z) = \boldsymbol{\alpha} \cdot \mathcal{C}(Z) + \mathcal{N}(0, \sigma) = \sum_{i=1}^{m} \alpha_i \mathcal{C}(Z)[i] + \mathcal{N}(0, \sigma) \ . \tag{8}$$

In the next proposition, we give an explicit formula of the SNR when an encoding function is involved to thwart SCA attacks.

Proposition 2. *Let $\mathcal{L}(Z)$ satisfy Eq.* (8)*. Then, for every Z in $\mathbb{F}_2^n$, the Signal-to-Noise Ratio satisfies:*

$$SNR = \left(\sum_{\substack{i,j=1 \\ i \neq j}}^{m} \mathsf{E}[\mathcal{C}(Z)[i]\mathcal{C}(Z)[j]] + (\sigma_\alpha^2 + 1) \sum_{i=1}^{m} \mathsf{E}[\mathcal{C}(Z)[i]] - \left(\sum_{i=1}^{m} \mathsf{E}[\mathcal{C}(Z)[i]] \right)^2 \right) / \sigma^2 \ . \tag{9}$$

Proof. The proof of Proposition 4 is given in Appendix D. □

Using the result of Proposition 4 and Eq. (6), one can evaluate the amount of traces required to reach a 90% of success rate when an encoding is applied to protect a sensitive data. For the sake of comparison, we will also evaluate this metric for some well known countermeasures. We list hereafter the leakage functions we consider:

- Unprotected: $\mathcal{L}_{\text{unpro}}(Z) = \boldsymbol{\alpha} \cdot \mathcal{C}_{\text{unpro}}(Z) + \mathcal{N}(0, \sigma)$, where $\mathcal{C}_{\text{unpro}}$ is the identity function.
- Software dual-rail [14]: $\mathcal{L}_{\text{dual}}(Z) = \boldsymbol{\alpha} \cdot \mathcal{C}_{\text{dual}}(Z) + \mathcal{N}(0, \sigma)$, where $\mathcal{C}_{\text{dual}}$ is the dual-rail code, described in Appendix A.
- Software constant weight [24]: $\mathcal{L}_{\text{cstHW}}(Z) = \boldsymbol{\alpha} \cdot \mathcal{C}_{\text{cstHW}}(Z) + \mathcal{N}(0, \sigma)$, where $\mathcal{C}_{\text{cstHW}}$ is the (3,6)-code described in Appendix A.
- Our proposed customized encoding: $\mathcal{L}(Z)_{\text{cust}} = \boldsymbol{\alpha} \cdot \mathcal{C}_{\text{cust}}(Z) + \mathcal{N}(0, \sigma)$, where $\mathcal{C}_{\text{cust}}$ is the code generated using Algorithm 1 for different codeword lengths.

In the sequel, we consider that the sensitive variable Z is a 4-bit variable, (*i.e.* $n = 4$). Then, for each of the above described leakage functions, we have computed the SNR over a set of 5.000 independent experiments using the result of Proposition 4. The standard deviation of the bit leakage weights σ_α was fixed at 0.25 and 0.5. Finally, we have deduced the $N_{90\%}$ using Eq. (6).

In Fig. 1, we plot the number of traces to achieve a success rate of 90% according to the noise standard deviation σ. For our customized encoding functions, we show the results for different codewords lengths, *i.e.* $\mathcal{C}_{\text{cust}} : \mathbb{F}_2^4 \mapsto \mathbb{F}_2^m$ with m in $[\![5, 10]\!]$.

From Fig. 1, the following observations could be emphasized:

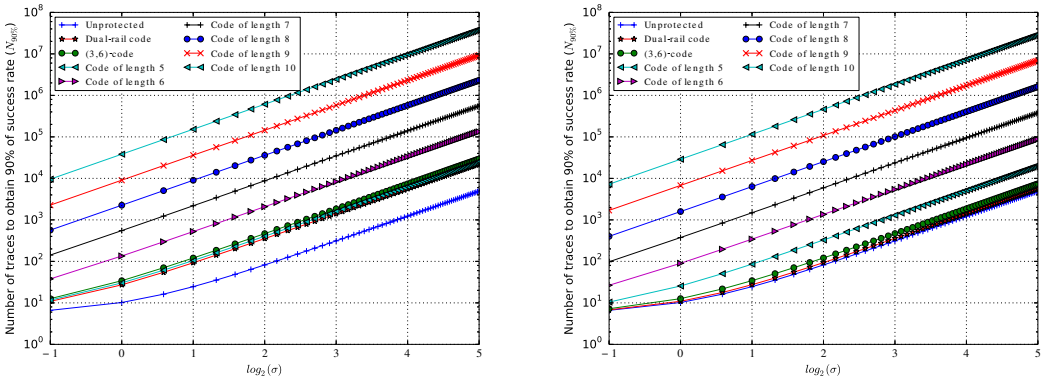


Fig. 1. Evolution of the number of traces to achieve a success rate of 90% (y-axis) according to an increasing noise standard deviation σ (x-axis in log scale base 2). Left: for $\sigma_\alpha = 0.25$. Right: for $\sigma_\alpha = 0.5$.

- As expected the constant weight encoding countermeasures are less efficient than our customized encoding functions. For instance, when the noise standard deviation equals 16, about 10.000 and 2.000 are sufficient to reach a success rate of 90% when σ_α equals 0.25 and 0.5 respectively. This is due to the fact that these codes are generated independently of the physical leakage by simply assuming a Hamming weight leakage model.
- The longer the code is, the more resistant the countermeasure is. In fact, our Algorithm 1 generates an encoding function such that the delta consumption of the selected codewords, the corresponding physical leakage variance, and the SNR are the lowest among all the subsets of codewords. So, the longer the code is, the more efficient our proposed algorithm is in selecting an encoding function that minimizes further the SNR. [4] For instance, when $\sigma_\alpha = 0.25$ and $\sigma = 2$, the SNR decreases from about 0.032 for the (3,6)-code to 7.8×10^{-5} for the customized code of length 10.
- For a fixed noise standard deviation, one can notice that if σ_α increases, the adversary will need less traces to achieve a success rate of 90%. For instance, when $\sigma = 8$ and the customized encoding of length 6 is used, the $N_{90\%}$ equals approximately 10.000 and 8.000 traces when σ_α varies from 0.25 to 0.5 as shown in Fig. 1. This observation is in-line with Eq. (D.1). In fact, when σ_α increases, the SNR increases accordingly and hence the $N_{90\%}$ decreases. To sum up, the degree of randomness of the leakage function has a noticeable impact on the amount of traces required by an adversary to achieve a success rate of 90%. So, the higher σ_α is, the longer encoding function a designer should use.

[4] In Appendix E, we provide some examples of the SNR computation.

– It is noteworthy that the code of length 5 is less efficient than the state-of-the-art countermeasures when $\sigma_\alpha = 0.25$. However, when $\sigma_\alpha = 0.5$, this customized code achieves a better result than the dual-rail and the (3,6)-code.

To conclude, our proposed encodings bring an overwhelming gain in terms of number of traces to succeed the CPA attack. For instance, to break the code of length 7, an adversary requires about 12 and 50 times more traces to achieve a CPA success rate of 90% compared to the dual-rail countermeasure when σ_α equals 0.25 and 0.5 respectively.

5 Security Evaluation of the New Customized Encoding

As argued on the evaluation framework introduced in [25], the robustness of a countermeasure encompasses two dimensions: its amount of leakage irrespective of any attack strategy and its resistance to specific attacks. So, the evaluation of protected implementations should hold in two steps. First, an information theoretic analysis determines the actual information leakage. Second, a security analysis determines the efficiency of various attacks in exploiting this leakage.

Following this evaluation framework, we start with an information theoretic analysis in the following subsection.

5.1 Information Theoretic Analysis

To evaluate the information revealed by our proposed encoding functions, we compute the *Mutual Information Metric* (MIM) between the sensitive variable Z and the leakage function: $\mathsf{I}[\mathcal{L}_{\text{cust}}(Z); Z] = \mathsf{H}[\mathcal{L}_{\text{cust}}(Z)] - \mathsf{H}[\mathcal{L}_{\text{cust}}(Z) \mid Z]$, where $\mathsf{H}[.]$ denotes the entropy function. For the sake of comparison, we evaluate the MIM for the leakage functions listed in Sec. 4.3 as well. Besides, we compute this metric also for a first-order masking leakage function:

$$\mathcal{L}_{\text{mask}}(Z) = (\boldsymbol{\alpha_1} \cdot (Z \oplus M)) \times (\boldsymbol{\alpha_2} \cdot M) + \mathcal{N}(0, \sigma) \ , \tag{10}$$

where M denotes a random mask defined over $\mathbb{F}_2^4$ and ($\boldsymbol{\alpha_1}$, $\boldsymbol{\alpha_2}$) are respectively the bit leakage weight vector of the masked data ($Z \oplus M$) and the mask (M) such that $\boldsymbol{\alpha_1} \neq \boldsymbol{\alpha_2}$. Put differently, we assume that the masked data bits and the mask bits leak independently[5].

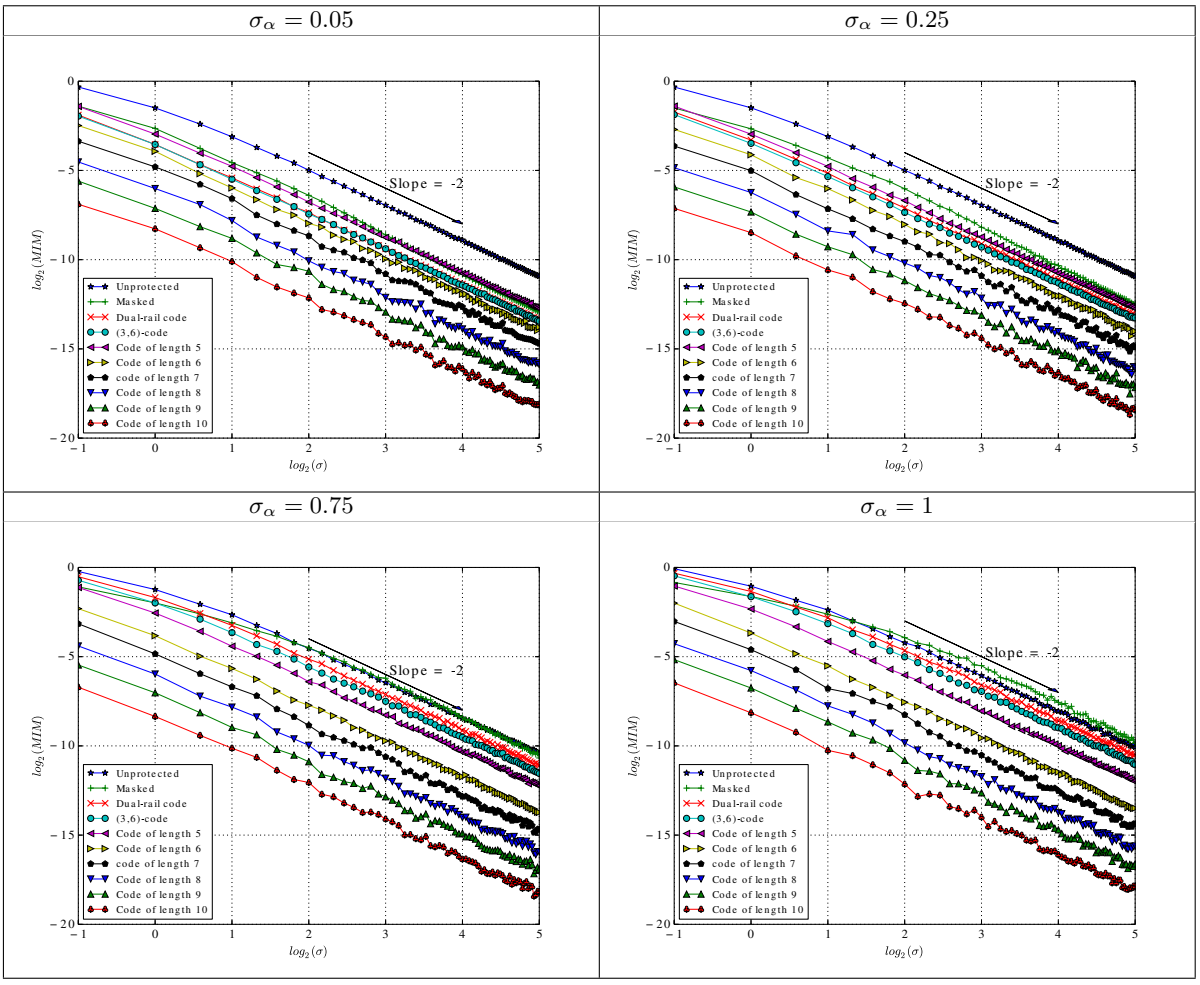
From Eq. (10), one can conclude that for masking we consider a *bivariate* leakage, *i.e.* a product combination of the two leakages (the masked data and the mask) is exploited by the adversary.

For each leakage function, the MIM was computed for several standard deviations of the bit leakage weights (σ_α in $\{0.05, 0.25, 0.75, 1\}$) and over a set of 200 independent experiments. The MIM is computed via numerical integration (Sec.4.1.b of [10]). This method is accurate when the leakage is mathematically generated to perform simulations. The obtained results are shown in Tab. 1.

From Tab. 1, the following observations can be emphasized:

[5] Our goal here is to analyze the masking countermeasure in the worst case scenario (*i.e.* the mask register and the masked data register have different leakage functions).

Table 1. Evolution of the MIM (y-axis in log scale base 2) according to an increasing noise standard deviation σ (x-axis in log scale base 2).



– Interestingly, all MIM curves are parallel and have the same slope (-2). In fact, it has been demonstrated that the mutual information is proportional to $\sigma^{(-2d)}$ for large enough noises, where d denotes the order of the smallest statistical moment in the leakage distribution depending on the secret key and corresponds also to the number of shares used to represent the sensitive data [4,11,26]. Since for all the considered leakage functions the sensitive data is represented with a single share (*i.e.* $d = 1$)[6], then the corre-

[6] For the masking leakage function, we stress the fact that we have used one share which corresponds to the product combination of the masked data share and the mask share (*i.e.* a second-order analysis of the first-order masking).

sponding mutual information decrease exponentially following a curve with slope (-2) when the noise standard deviation σ increases. As expected, this confirms that the unprotected implementation and the encoding-based countermeasures lead to first-order univariate weaknesses and that the masking countermeasure leads to first-order bivariate weakness.
- Despite having the same gradient, the amount of information leaked differs from a leakage function to another. For instance, one can see that whatever the σ_α value is, our proposed encoding functions of length superior to 6 leak less than the other encoding countermeasures and the first-order masking. This result is in-line with that of Sec. 4.3. In fact, the longer the code is, the less information is leaked, the lowest the SNR is, and the more traces are needed to break the implementation.
- For $\sigma_\alpha \leq 0.25$, our customized code of length 5 performs worse since it leaks more than the dual-rail and the (3,6)-code. This result is also in-line with that shown in Fig. 1. In fact, an adversary requires less traces to break the optimal code of length 5 than the (3,6)-code. This could be simply explained by the fact that for small σ_α the best code is a constant weight one and no such a code exists for length 5 to generate 16 codewords.
- It is noteworthy that the first-order masking performs worse when $\sigma_\alpha \geq 0.75$. It leaks slightly more information than an unprotected implementation. This result can be explained by the fact that when the bits of the two shares (the masked data and the mask) leak "very" differently, the countermeasure is doubly impacted (*i.e.* unbalance of the masked data leakage and unbalance of the mask leakage). This implies that the security guarantee by masking vanish in such a scenario. This result is in-line with that obtained in [11], where the MIM has been evaluated when the masking and the unprotected leakage functions radically deviate from the idealized Hamming weight model.
- It appears also that the degree of the deviation from the Hamming weight model (*i.e.* σ_α) has a noticeable influence on the amount of information leaked. In fact, for a fixed noise standard deviation σ, the higher σ_α, the larger the leakage. The same observation has been pinpointed in [11], *i.e.* the quantity of information leaked is strongly affected by the degree of randomness of the leakage function. Moreover, this result is in-line with that discussed in Sec. 4.3, *i.e.* the higher σ_α, the less number of traces are needed to achieve a 90% of success rate.

5.2 Side-Channel Security Analysis

To complete the security evaluation of our proposal, we conduct in this subsection a security analysis to evaluate its resistance to thwart SCA attacks. Namely, we perform a security evaluation of the stochastic attacks, for which a strong consistency with the previous security metrics analyses (*i.e.* the information theoretic analysis, the SNR and the MTD) should hold. To do so, we detail hereafter the attack simulation setup.

Simulation Setup. The leakage measurements have been simulated as samples of all the leakage functions listed in Sec. 4.3 and that detailed in Eq. (10)

for the first-order masking countermeasure. Moreover, the sensitive variable Z was chosen to be a PRESENT Sbox output of the form $S(X \oplus k)$, where X represents a varying 4-bit plaintext and k represents the key nibble to recover.

Attack scenarios. For our simulation attacks, we focus on two scenarios:

- *The best-case scenario:* we consider a powerful adversary who has access to the bit leakage weights and the characteristics of the optimal used code (*i.e.* the code length and the subset of the codewords). Then, he performs a stochastic attack by targeting the protected variable $\mathcal{C}(S(X \oplus k))$.
- *The worst-case scenario:* we consider a more realistic (and much weaker) adversary who has only the control on the target device to characterize the physical leakage. However, the characteristics of the used code are unknown. So, the adversary performs a linear regression over a 4-bit variable, *i.e.* the PRESENT Sbox output $S(X \oplus k)$.

For each scenario and for each leakage function, we compute the success rate of the stochastic attack [16] over 200 independent experiments. Moreover, this security metric was computed for several standard deviations of the bit leakage weights (σ_α in $\{0.05, 0.25, 0.75, 1\}$). The noise standard deviation was fixed at $\sigma = 0.25$. The simulation results in the best-case scenario are plotted in Tab. 2.[7]

Simulation results. For the best-case scenario, the results shown in Tab. 2 are in-line with those obtained during the information theoretic evaluation. In fact, when the $\sigma_\alpha \leq 0.25$, the optimal code of length 5 performs worse since an adversary requires less traces to achieve a 100% of success rate than the constant weight countermeasures. Besides, we conclude again that the longer the code is, the more resistant the implementation is. Moreover, the higher the standard deviation of the bit leakage weights is, the less efficient the encoding function is.

For the worst-case scenario, as expected, the stochastic attack performs worse since the adversary does not have the control on the code length and the subset of codewords used for the protection. So, the profiling phase outputs an imprecise leakage model which impacts the attack efficiency.

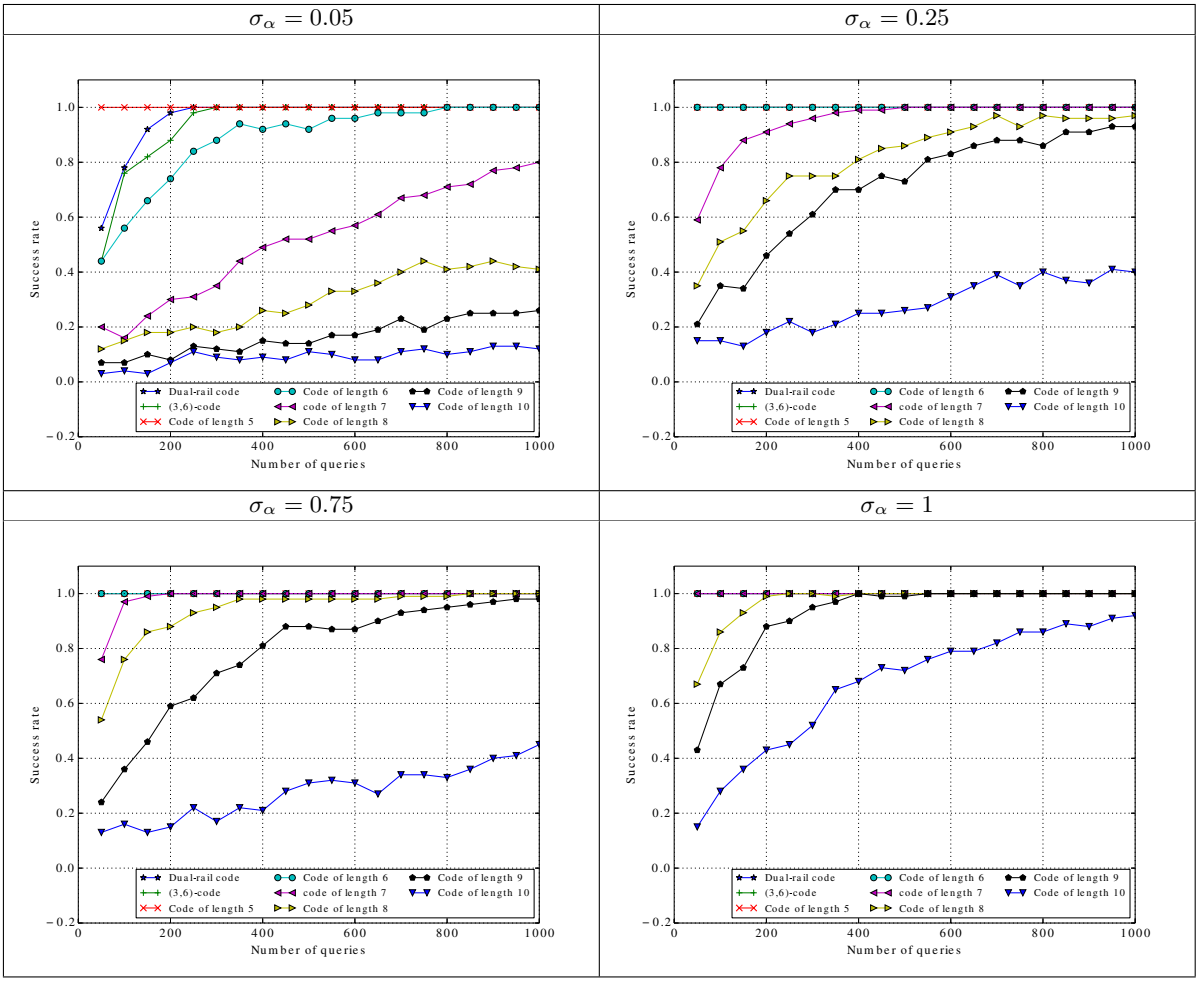
These simulation results also highlight the inefficiency of fixed constant weight codes such as the dual-rail in all the presented models. Customized encodings of the same length of 8 bits exhibit a much higher resistance.

6 Practical Evaluation of the New Customized Encoding

In the previous sections, we have confronted our theoretical analyses based on the SNR and the MTD security metrics with simulations based on the security evaluation framework proposed in [25]. In the following, we aim to confront these results against real measurements.

[7] The simulation results in the worst-case scenario can be found in Appendix F.

Table 2. Stochastic attack results in the best-case scenario for a noise standard deviation $\sigma = 0.25$.



6.1 Implementation considerations and memory-security trade-off

Encoding of sensitive data with codewords of longer length, *e.g.* representing the PRESENT Sbox output nibble as a byte, seems unreasonable for embedded software products at first, as the computation tables grow quadratically in size with the length of the code. In order to avoid large memory penalties when implementing our solution, a trick detailed in [14] and [24] has to be used. It consists in encoding a n-bit variable as two separate halves. This way, the linear and non-linear operations of a block-cipher can be performed at a much lower memory cost than with a double-length encoding.

We begin with a quick reminder of this trick. Listing 1.1 shows how to perform an encoded memory access with an 8-bit input, encoded in two words of 7 bits each. This kind of operation could be an AES Sbox or a XOR operation between two nibbles for example.

```
1  // R3 = @table_msb, R4 = @table_lsb,  R5 = @shift1_table, R6 =
       @shift7_table
2  // R0 = operand MSB = 0xxxxxxx, R1 = operand LSB = 0yyyyyyy
3  LDRB    R2, [R5,R0]          // R2 = 00000000xxxxxxx0
4  EOR     R0,R0,R0             // Clear R0
5  LDRH    R0, [R6,R2]          // R0 = 00xxxxxxx0000000
6  EOR     R0,R0,R1             // R0 = 00xxxxxxyyyyyyy
7  EOR     R1,R1,R1             // Clear R1
8  EOR     R2,R2,R2             // Clear R2
9  LDRB    R1, [R3,R0]          // R1 =table_msb[operand] (7 bits)
10 LDRB    R2, [R4,R0]          // R2 =table_lsb[operand] (7 bits)
```

Listing 1.1. Double-length encoded access for a code of length 7 (ARM assembly)

This procedure works if we assume all registers leak more or less the same function of their inputs.[8] If registers happen to leak vastly differently, then we need different encodings for each register and code conversions between memory accesses. This study is out of the scope of this paper.

As the code for the most significant bits of a register could be different than the one for the least significant bits, we require to tabulate the shift operation so that it outputs the correct code for the given destination within the register (*shift1_table* and *shift7_table* in Listing 1.1). In the end this shows all classical operations of a block cipher (e.g. XOR, Sbox) can be covered. Regarding bit-level operations (e.g. permutation in DES, PRESENT), a solution may consist in tabulating these operations too, if there is enough memory available. As shown in Fig. 2, inside the look-up table we decode the inputs $\mathcal{C}(x_i)$ by computing $(\mathcal{C}^{-1})$, apply the bit-level permutation (P) and encode the result $(Y_i = P(x_i))$.

In the former case, the overhead compared to an unprotected implementation would be the same as the one obtained in [24]. This means that an encoded AES would execute only roughly 4 times slower than its unprotected version.

Regarding the choice of the code length, it is up to the designer to choose the suitable length that guarantees the best performance-security trade-off according to the perceived physical leakage. Perhaps a recommendation could be to estimate the minimum number of traces to disclose the key (MTD) for different code lengths (as investigated in Sec. 4.3) then select the encoding function according to the available memory and the required level of security.

[8] It was the case for our practical experiment as detailed in Appendix J.

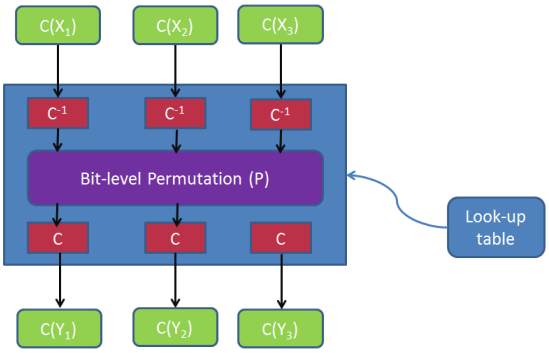


Fig. 2. Protecting bit-level permutation with encoding functions.

6.2 Experimental Setup

We have performed several practical experiments using a Micro-Controller integrated circuit from STMicroelectronics. Namely, we choose the $STM32F3$ circuit [27] based on the 32-bit RISC ARM Cortex-M4F processor core with 90nm CMOS process. In order to assess the practicability of our new framework in realistic case, we use 4 different copies of the $STM32F3$ circuit (referred as copy #1, #2, #3 and #4). Our goal is to provide an answer to the following question:

Does a customized encoding for one circuit ensure the same security level when implemented on a different circuit of the same family?

So, the idea behind using four copies is to enable us to apply our framework (*i.e.* stochastic profiling of the leakage and customized encoding generation) on one copy and to use the same encoding functions to protect the other copies without a prior profiling. The target operation is a PRESENT Sbox computation protected by a customized encoding function with different codeword lengths. The side-channel traces were obtained by measuring the electromagnetic radiations (EM) emitted by the target device.

6.3 Attack Experiments and Results

To perform our profiling phase, we have first acquired 25.000 EM traces recording an AES Sbox computation when implemented on copy #1. The use of the AES Sbox (8 bits output) rather than the PRESENT Sbox (4 bits output) was necessary to extract the weights of all the 8 bits of a register by a stochastic approach. To do so, we have performed a linear regression attack and we have captured the averaged bit weights returned when the attack succeeds to find key[9]. The

[9] As detailed in [16], the stochastic attack does not only return the best key candidate but also a linear regression of the leakage. We computed the average of the bit leakage coefficients during a time window where the attack succeeds.

obtained bit weights α_i are plotted in Appendix G. For this circuit, we have observed that it leaks closely to the Hamming weight model which implies an exploitable penalty in the security of constant weight countermeasures.

Second, we have executed Algorithm 1 to obtain the optimal encoding functions of length varying in $[\![5, 7]\!]$ to protect the 4-bit PRESENT Sbox output. Third, for each code length, we have implemented the protected PRESENT Sbox on each copy. We stress the fact that we have used the obtained encoding functions (for copy #1) to protect the three other copies without a prior leakage characterization. For fair comparison, 50.000 EM traces [10] were acquired with a fixed experimental setup: *i.e.* the same electromagnetic probe, the same probe's position, the same oscilloscope configuration to sample the measurements and the same temporal acquisition window. The code setup is a simple Sbox access in RAM which overwrites a register containing zero. The Sbox was aligned in memory for every encoding, and the same registers were used for each circuit copies.[11]

Finally, we conducted 10 independent *enhanced CPA attacks*[12] against each implementation of the four copies (*i.e.* we used 10 independent set of 5.000 EM traces). The evolution of the averaged rank of the correct key among 16 (4-bit keys) is plotted in Fig. 3 for each circuit and code length.

The various encodings perform as expected on all circuits, although they were constructed from the profiling of only one of them. Longer codes provide higher resistance, but only very slightly for a code of length 5. These practical results are in-line with the simulation ones shown on Sec. 5.2. Overall, these results confirm that one can profile a single device, devise the corresponding encodings, then use them to protect another device of the same family without loosing much in resistance. More studies should be performed in order to fully assess the generic side of the countermeasure nonetheless.

7 Conclusion

In this paper, we have proposed a new framework for building customized encoding function according to the physical leakage characteristics of the target device. It gives assurance that even under good profiling conditions for an attacker, the Signal-to-Noise Ratio is close to minimal. We also showed how much leakage reduction is to be expected for previous constant weight countermeasures in the case of an imbalanced leakage. The security evaluation conducted has shown the overwhelming advantages of our proposal compared to the existing constant weight countermeasures in more realistic scenarios. It is also more difficult to attack than a first-order masking when the latter's shares can be easily combined by an attacker. It is also possible to obtain the same performance impact

[10] An example of an EM trace is provided in Appendix H.

[11] The assembly code used is detailed in Appendix I.

[12] We assume a powerful adversary who has access to the used encoding function $\mathcal{C}_{\text{cust}}$ and the bit leakage weight vector $\boldsymbol{\alpha}$. Hence, he is able to compute $\rho[\mathcal{L}(Z); \boldsymbol{\alpha} \cdot \mathcal{C}_{\text{cust}}(Z)]$. We emphasize the fact each attack was processed on the whole trace.

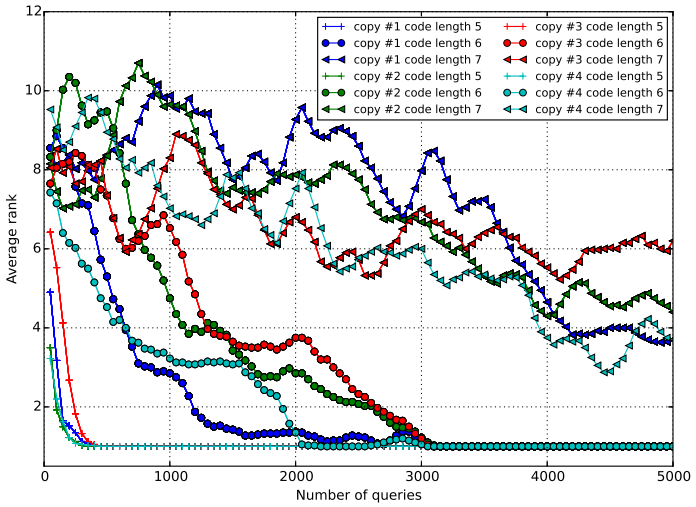


Fig. 3. Evolution of the correct key rank according to the number of observations.

as constant weight implementations, making customized encodings faster than known second-order masking schemes. Besides, the obtained results within the four considered security metrics (*i.e.* the SNR, the MTD, the information theoretic, the success rate of stochastic attack) are in-line, proving the tightness of our security evaluation process. Finally, the practical assessment of our solution have enabled us to confirm its practicability to protect cryptographic operations when applied on four different copies of the same device.

Our work opens avenues for further research of new encoding functions when assuming a higher-order leakage model (*e.g.* quadratic, cubic, ...) and also the study of new designs combining both masking and encodings. Another future work will consist in studying the inter-conversion of encoding functions when the registers of a circuit have different leakage model and then, several customized codes have to be used to protect a block cipher.

Acknowledgments

This work has been partially funded by the ANR project SERTIF. We thank anonymous reviewers of FSE 2016 for the various constructive comments and suggestions.

References

1. L. Batina, B. Gierlichs, and K. Lemke-Rust. Differential Cluster Analysis. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 112–127, Lausanne,CH, 2009. Springer-Verlag.
2. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *CHES*, volume 4727 of *LNCS*, pages 450–466. Springer, September 10-13 2007.
3. É. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, August 11–13 2004. Cambridge, MA, USA.
4. C. Carlet, J.-L. Danger, S. Guilley, H. Maghrebi, and E. Prouff. Achieving side-channel high-order correlation immunity with leakage squeezing. *J. Cryptographic Engineering*, 4(2):107–121, 2014.
5. S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, August 2002. San Francisco Bay, USA.
6. C. Chen, T. Eisenbarth, A. Shahverdi, and X. Ye. *Balanced Encoding to Mitigate Power Analysis: A Case Study*. 2015.
7. Z. Chen and Y. Zhou. Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage. In *CHES*, volume 4249 of *LNCS*, pages 242–254. Springer, October 10-13 2006.
8. S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot. A White-Box DES Implementation for DRM Applications. In *Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002*, volume 2696 of *LNCS*, pages 1–15. Springer, 2002.
9. J.-S. Coron. Higher Order Masking of Look-Up Tables. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2014.
10. J. Doget, E. Prouff, M. Rivain, and F.-X. Standaert. Univariate side channel attacks and leakage modeling. *J. Cryptographic Engineering*, 1(2):123–144, 2011.
11. A. Duc, S. Faust, and F. Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In *EUROCRYPT 2015, Sofia, Bulgaria, April 26-30*, pages 401–429, 2015.
12. L. Genelle, E. Prouff, and M. Quisquater. Thwarting higher-order side channel analysis with additive and multiplicative maskings. In *CHES 2011*.
13. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In *CHES, 10th International Workshop*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, August 10-13 2008. Washington, D.C., USA.
14. P. Hoogvorst, J.-L. Danger, and G. Duc. Software Implementation of Dual-Rail Representation. In *COSADE*, February 24-25 2011. Darmstadt, Germany.
15. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO*, volume 1666 of *LNCS*, pages pp 388–397. Springer, 1999.
16. V. Lomné, E. Prouff, and T. Roche. Behind the scene of side channel attacks. In K. Sako and P. Sarkar, editors, *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 506–525. Springer, 2013.
17. S. Mangard. Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness. In *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004. San Francisco, CA, USA.

18. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006.
19. É. Peeters, F.-X. Standaert, N. Donckers, and J.-J. Quisquater. Improved Higher-Order Side-Channel Attacks With FPGA Experiments. In *CHES*, volume 3659 of *LNCS*, pages 309–323. Springer-Verlag, 2005. Edinburgh, UK.
20. E. Prouff, M. Rivain, and R. Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
21. M. Rivain and E. Prouff. Provably Secure Higher-Order Masking of AES. In *CHES*, pages 413–427, 2010.
22. W. Schindler. Advanced stochastic methods in side channel analysis on block ciphers in the presence of masking. *Journal of Mathematical Cryptology*, 2(3):291–310, October 2008. ISSN (Online) 1862-2984, ISSN (Print) 1862-2976, DOI: 10.1515/JMC.2008.013.
23. W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In LNCS, editor, *CHES*, volume 3659 of *LNCS*, pages 30–46. Springer, Sept 2005. Edinburgh, Scotland, UK.
24. V. Servant, N. Debande, H. Maghrebi, and J. Bringer. Study of a novel software constant weight implementation. In *CARDIS, Paris, France, November 5-7,*, pages 35–48, 2014.
25. F.-X. Standaert, T. Malkin, and M. Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *EUROCRYPT*, volume 5479 of *LNCS*, pages 443–461. Springer, April 26-30 2009. Cologne, Germany.
26. F.-X. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The World is Not Enough: Another Look on Second-Order DPA. In *ASIACRYPT*, volume 6477 of *LNCS*, pages 112–129. Springer, December 5-9 2010.
27. STMicroelectronics designer. `http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1576?icmp=ss1576_pron_pr_jan2015&sc=stm32f3-pr3`.
28. N. Veyrat-Charvillon and F.-X. Standaert. Mutual Information Analysis: How, When and Why? In *CHES*, volume 5747 of *LNCS*, pages 429–443. Springer, September 6-9 2009. Lausanne, Switzerland.

A The Dual-Rail and (3,6)-Codes

The dual-rail code and the (3,6)-code are provided in Tab. 3 and Tab. 4 respectively.

Table 3. The dual-rail code.

$0 \rightarrow 01010101$	$4 \rightarrow 01100101$	$8 \rightarrow 10010101$	$12 \rightarrow 10100101$
$1 \rightarrow 01010110$	$5 \rightarrow 01100110$	$9 \rightarrow 10010110$	$13 \rightarrow 10100110$
$2 \rightarrow 01011001$	$6 \rightarrow 01101001$	$10 \rightarrow 10011001$	$14 \rightarrow 10101001$
$3 \rightarrow 01011010$	$7 \rightarrow 01101010$	$11 \rightarrow 10011010$	$15 \rightarrow 10101010$

Table 4. The (3,6)-code.

$0 \to 000111$	$4 \to 010011$	$8 \to 011010$	$12 \to 100110$
$1 \to 001011$	$5 \to 010101$	$9 \to 011100$	$13 \to 101001$
$2 \to 001101$	$6 \to 010110$	$10 \to 100011$	$14 \to 101010$
$3 \to 001110$	$7 \to 011001$	$11 \to 100101$	$15 \to 101100$

B Some Precomputed Values for the Quantiles of a Normal Distribution

We provide in Tab. 5 some precomputed values of the quantile of a normal distribution for the 2-sided confidence interval with error $(1-\beta)$.

Table 5. Some precomputed values for the quantiles of a normal distribution.

Confidence level [%] $(1-\beta)$	$Z_{1-\beta}$
60	0.842
80	1.282
90	**1.645**
95	1.960
98	2.326
99	2.576

C Proof of Proposition 1

We recall hereafter Proposition 1.

Proposition 3. *Let $\mathcal{L}(Z)$ satisfies $\mathcal{L}(Z) = \sum_{i=1}^{n} \alpha_i Z[i] + \mathcal{N}(0,\sigma)$. Then, the optimal correlation function satisfies:*

$$\rho_{opt} = \sqrt{\frac{1}{1+\frac{1}{SNR}}} \ , \tag{C.1}$$

where the SNR can be rewritten:

$$SNR = \left(\mathsf{Var}[\sum_{i=1}^{n} \alpha_i Z[i]] \right) / \sigma^2 \ .$$

Proof. On the one hand, we have:

$$\begin{aligned}
\mathsf{Var}[\mathsf{E}[\mathcal{L}(Z) \mid (Z=z)]] =& \mathsf{Var}[\mathsf{E}[\sum_{i=1}^{n} \alpha_i Z[i] + \mathcal{N}(0,\sigma) \mid (Z=z)]] \\
=& \mathsf{Var}[\mathsf{E}[\sum_{i=1}^{n} \alpha_i Z[i] \mid (Z=z)] + \mathsf{E}[\mathcal{N}(0,\sigma)]] \quad \text{(two independent variables)} \\
=& \mathsf{Var}[\sum_{z \in \mathbb{F}_2^n} \mathsf{P}[Z=z] \left(\sum_{i=1}^{n} \alpha_i z[i] \right) + 0] \\
=& \mathsf{Var}[\sum_{i=1}^{n} \alpha_i Z[i]] \ . \qquad \text{(C.2)}
\end{aligned}$$

On the other hand, we have:

$$\begin{aligned}
\mathsf{Var}[\mathcal{L}(Z)] =& \mathsf{Var}[\sum_{i=1}^{n} \alpha_i Z[i] + \mathcal{N}(0,\sigma)] \\
=& \mathsf{Var}[\sum_{i=1}^{n} \alpha_i Z[i]] + \mathsf{Var}[\mathcal{N}(0,\sigma)] \quad \text{(two independent variables)} \\
=& \mathsf{Var}[\sum_{i=1}^{n} \alpha_i Z[i]] + \sigma^2 \ . \qquad \text{(C.3)}
\end{aligned}$$

Hence, Eq. (C.2) and Eq. (C.3) together imply that the optimal correlation function can be expressed as:

$$\begin{aligned}
\rho_{\text{opt}} &= \sqrt{\frac{\mathsf{Var}[\mathsf{E}[\mathcal{L}(Z) \mid Z=z]]}{\mathsf{Var}[\mathcal{L}(Z)]}} \\
&= \sqrt{\frac{\mathsf{Var}[\sum_{i=1}^{n} \alpha_i Z[i]]}{\mathsf{Var}[\sum_{i=1}^{n} \alpha_i Z[i]] + \sigma^2}} \\
&= \sqrt{\frac{1}{1 + \frac{\sigma^2}{\mathsf{Var}[\sum_{i=1}^{n} \alpha_i Z[i]]}}} \\
&= \sqrt{\frac{1}{1 + \frac{1}{SNR}}} \ ,
\end{aligned}$$

which leads to Eq. (C.1) and achieves the proof. □

D Proof of Proposition 2

We recall hereafter Proposition 2.

Proposition 4. *Let $\mathcal{L}(Z)$ satisfy $\mathcal{L}(Z) = \boldsymbol{\alpha} \cdot \mathcal{C}(Z) + \mathcal{N}(0, \sigma)$ Then, for every Z in $\mathbb{F}_2^n$, the Signal-to-Noise Ratio satisfies:*

$$SNR = \left(\sum_{\substack{i,j=1 \\ i \neq j}}^{m} \mathsf{E}[\mathcal{C}(Z)[i]\mathcal{C}(Z)[j]] + (\sigma_\alpha^2 + 1) \sum_{i=1}^{m} \mathsf{E}[\mathcal{C}(Z)[i]] - \left(\sum_{i=1}^{m} \mathsf{E}[\mathcal{C}(Z)[i]] \right)^2 \right) / \sigma^2 \ . \tag{D.1}$$

Proof. Let $\mathcal{C} : \mathbb{F}_2^n \mapsto \mathbb{F}_2^m$ be an encoding function, then the SNR can be rewritten:

$$\begin{aligned} \text{SNR} &= \frac{\mathsf{Var}[\boldsymbol{\alpha} \cdot \mathcal{C}(Z)]}{\sigma^2} \\ &= \underbrace{\frac{\mathsf{E}[(\boldsymbol{\alpha} \cdot \mathcal{C}(Z))^2]}{\sigma^2}}_{\text{Term \#1}} - \underbrace{\frac{(\mathsf{E}[\boldsymbol{\alpha} \cdot \mathcal{C}(Z)])^2}{\sigma^2}}_{\text{Term \#2}} \ . \end{aligned}$$

This result is directly deduced from the formula of the variance. We can now derive the expression of the two terms. First of all, we have:

$$\begin{aligned} \text{Term \#1} =& \frac{1}{\sigma^2} \mathsf{E}[(\boldsymbol{\alpha} \cdot \mathcal{C}(Z))^2] \\ =& \frac{1}{\sigma^2} \mathsf{E}[\sum_{i,j=1}^{m} \alpha_i \alpha_j \mathcal{C}(Z)[i]\mathcal{C}(Z)[j]] \\ =& \frac{1}{\sigma^2} \mathsf{E}[\sum_{\substack{i,j=1 \\ i \neq j}}^{m} \alpha_i \alpha_j \mathcal{C}(Z)[i]\mathcal{C}(Z)[j] + \sum_{i=1}^{m} \alpha_i^2 \mathcal{C}(Z)[i]] \\ =& \frac{1}{\sigma^2} \mathsf{E}[\sum_{\substack{i,j=1 \\ i \neq j}}^{m} \alpha_i \alpha_j \mathcal{C}(Z)[i]\mathcal{C}(Z)[j]] + \mathsf{E}[\sum_{i=1}^{m} \alpha_i^2 \mathcal{C}(Z)[i]] \\ =& \frac{1}{\sigma^2} \sum_{\substack{i,j=1 \\ i \neq j}}^{m} \mathsf{E}[\alpha_i \alpha_j] \mathsf{E}[\mathcal{C}(Z)[i]\mathcal{C}(Z)[j]] + \sum_{i=1}^{m} \mathsf{E}[\alpha_i^2] \mathsf{E}[\mathcal{C}(Z)[i]] \quad (\alpha_i \text{ and } \mathcal{C}(Z)[i] \text{ are independent}) \\ =& \frac{1}{\sigma^2} \sum_{\substack{i,j=1 \\ i \neq j}}^{m} \mathsf{E}[\alpha_i] \mathsf{E}[\alpha_j] \mathsf{E}[\mathcal{C}(Z)[i]\mathcal{C}(Z)[j]] + \sum_{i=1}^{m} \mathsf{E}[\alpha_i^2] \mathsf{E}[\mathcal{C}(Z)[i]] \quad (\alpha_i \text{ are independent}) \\ =& \frac{1}{\sigma^2} \sum_{\substack{i,j=1 \\ i \neq j}}^{m} \mathsf{E}[\mathcal{C}(Z)[i]\mathcal{C}(Z)[j]] + (\sigma_\alpha^2 + 1) \sum_{i=1}^{m} \mathsf{E}[\mathcal{C}(Z)[i]] \ . \quad (\mathsf{E}[\alpha_i] = 1 \text{ and } \mathsf{E}[\alpha_i^2] = \sigma_\alpha^2 + 1) \end{aligned} \tag{D.2}$$

Then, we evaluate the Term #2:

$$\begin{aligned}
\text{Term } \#2 &= \frac{\left(\mathsf{E}[\boldsymbol{\alpha} \cdot \mathcal{C}(Z)]\right)^2}{\sigma^2} \\
&= \frac{\left(\mathsf{E}[\sum_{i=1}^{m} \alpha_i \mathcal{C}(Z)[i]]\right)^2}{\sigma^2} \\
&= \frac{\left(\sum_{i=1}^{m} \mathsf{E}[\alpha_i]\mathsf{E}[\mathcal{C}(Z)[i]]\right)^2}{\sigma^2} \\
&= \frac{\left(\sum_{i=1}^{m} \mathsf{E}[\mathcal{C}(Z)[i]]\right)^2}{\sigma^2} \ . \qquad \text{(D.3)}
\end{aligned}$$

Hence, Eq. (D.2) and Eq. (D.3) lead to Eq. (D.1) and achieve the proof. □

E Examples of Some Customized Encoding Functions and the Corresponding SNR Values

To evaluate the SNR, we have generated at random two 10-bit leakage weight vectors $\boldsymbol{\alpha_1}$ and $\boldsymbol{\alpha_2}$, according to a standard deviation σ_α equals 0.25 and 0.5. The returned weights of each bit are plotted in Fig. 4. We also plotted in green dashed line the value corresponding to the bit leakage weight in a Hamming weight model, *i.e.* $\alpha_i = 1$ for all i in $[\![1, 10]\!]$.

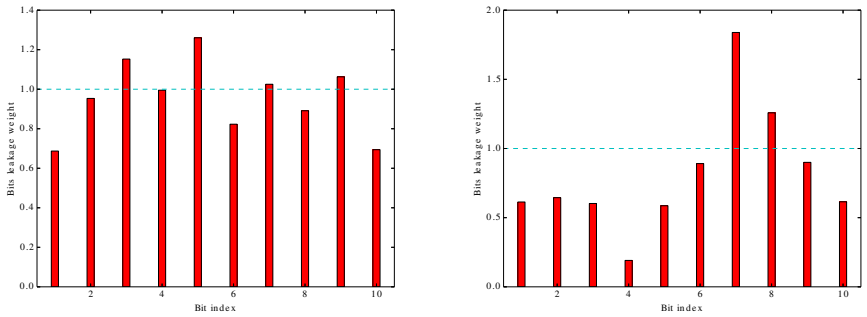


Fig. 4. Bit leakage weights obtained using linear regression. Left: for $\sigma_\alpha = 0.25$. Right: for $\sigma_\alpha = 0.5$.

Then, for each bit leakage weight vectors, we have computed the optimal code using Algorithm 1. Finally, we have deduced the corresponding codewords delta

consumption, the variance and the SNR for a fixed noise standard deviation $\sigma = 2$. The obtained results are highlighted in Tab. 6 and Tab. 7 for σ_α equals 0.25 and 0.5 respectively.

Table 6. SNR computation when $\sigma_\alpha = 0.25$.

$\boldsymbol{\alpha_1}$	[0.687119, 0.953471, 1.152647, 0.994985, 1.260224, 0.822996, 1.025128, 0.891029, 1.062778, 0.694017]										
Code length	Codewords used								Delta power consumption	Power consumption variance	SNR
5	0x14	0x11	0xa	0xc	0x6	0x9	0x3	0x5	1.368913	0.203826	0.101913
	0x1a	0x1c	0x16	0x19	0x13	0x15	0xe	0xb			
6	0x34	0x29	0x23	0x15	0x38	0x2c	0x32	0x19	0.600292	0.031336	0.015668
	0x26	0xd	0x13	0x7	0x2a	0x1c	0x16	0xb			
7	0x58	0xb	0x51	0x64	0x32	0x4c	0x1a	0x45	0.296494	0.009032	0.004516
	0x29	0x13	0x26	0xe	0x54	0x38	0x7	0x31			
8	0xc9	0x8e	0x56	0xe2	0x65	0x99	0xb2	0x35	0.135876	0.002258	0.001129
	0x8b	0x53	0x27	0xd8	0xac	0x74	0xca	0x4d			
9	0x6a	0x116	0x8d	0xa6	0x4e	0x1b0	0x161	0x113	0.067792	0.000602	0.000301
	0xa3	0x2d	0x158	0xe8	0x194	0x145	0x9a	0x4b			
10	0x2ca	0x174	0xd6	0xf1	0x22e	0x14e	0x169	0xcb	0.032354	0.000156	7.8×10^{-5}
	0x2f	0x3a4	0x2b8	0x1d8	0x1a5	0x13c	0x9e	0xb9			
8 (*the dual-rail code*)	0x55	0x56	0x59	0x5a	0x65	0x66	0x69	0x6a	1.097750	0.092498	0.046249
	0x95	0x96	0x99	0x9a	0xa5	0xa6	0xa9	0xaa			
6 (*the (3,6)-code*)	0x7	0xb	0xd	0xe	0x13	0x15	0x16	x19	0.940088	0.064739	0.032369
	0x1a	0x1c	0x23	0x25	0x26	0x29	0x2a	0x2c			

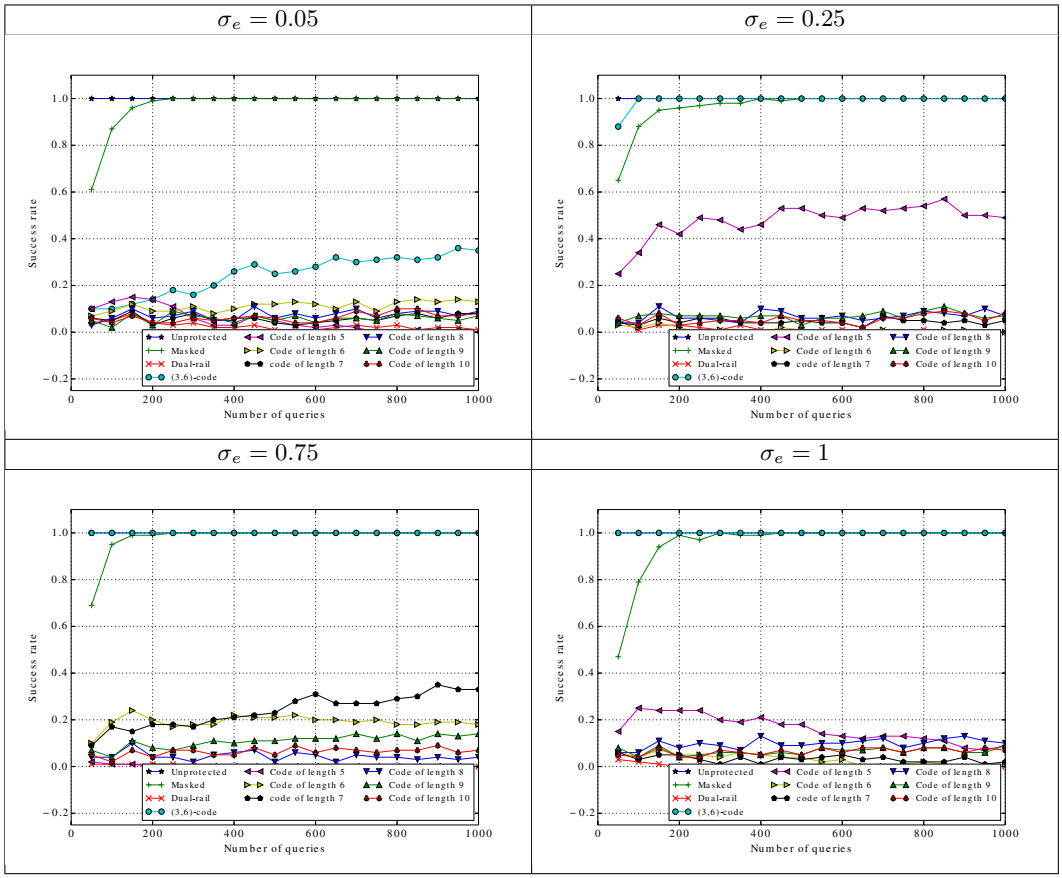
Table 7. SNR computation when $\sigma_\alpha = 0.5$.

$\boldsymbol{\alpha_2}$	[0.613331, 0.644584, 0.602531, 0.190986, 0.586268, 0.890951, 1.838814, 1.257943, 0.899922, 0.614699]										
Code length	Codewords used								Delta power consumption	Power consumption variance	SNR
5	0x5	0x11	0x14	0x9	0xc	0x18	0x7	0x13	0.671647	0.057365	0.028682
	0x16	0xb	0xe	0x1a	0x15	0xd	0x19	0x1c			
6	0x2e	0x1e	0x36	0x3c	0xb	0x23	0x29	0x13	0.346735	0.013739	0.006869
	0x19	0x31	0xf	0x27	0x2d	0x17	0x1d	0x35			
7	0x3c	0x9	0x6c	0x78	0x16	0x46	0x52	0x26	0.315482	0.013035	0.006517
	0x32	0x62	0x1e	0x4e	0x5a	0x2e	0x3a	0x6a			
8	0xda	0x1b	0xf2	0x33	0xf9	0x93	0x2e	0x8e	0.113723	0.001548	0.000774
	0x53	0xa6	0xad	0x4e	0x66	0x6d	0xc6	0x7			
9	0x14c	0x15a	0x145	0x153	0x9c	0x95	0xcc	0xda	0.056420	0.000310	0.000155
	0xc5	0x18c	0xe	0xd3	0x19a	0x185	0x7	0x193			
10	0xb4	0x1b1	0x8a	0x218	0x19	0x3a2	0xa6	0x1a3	0.021138	4.03×10^{-5}	2.01×10^{-5}
	0x234	0x331	0x35	0x20a	0xb	0x226	0x323	0x27			
8 (*the dual-rail code*)	0x55	0x56	0x59	0x5a	0x65	0x66	0x69	0x6a	1.582321	0.170241	0.085120
	0x95	0x96	0x99	0x9a	0xa5	0xa6	0xa9	0xaa			
6 (*the (3,6)-code*)	0x7	0xb	0xd	0xe	0x13	0x15	0x16	x19	1.895789	0.310569	0.155284
	0x1a	0x1c	0x23	0x25	0x26	0x29	0x2a	0x2c			

F Security Evaluation results of the Worst-Case Scenario

The simulation results obtained in the worst-case scenario are provided in Tab. 8.

Table 8. Stochastic attack results in the worst-case scenario for a noise standard deviation $\sigma = 0.25$.



G Linear Regression Bit Weights Characterized on Copy #1 of the STM32F3

From Fig. 5, one can conclude that the STM32F3 circuit leaks closely to the Hamming weight model.

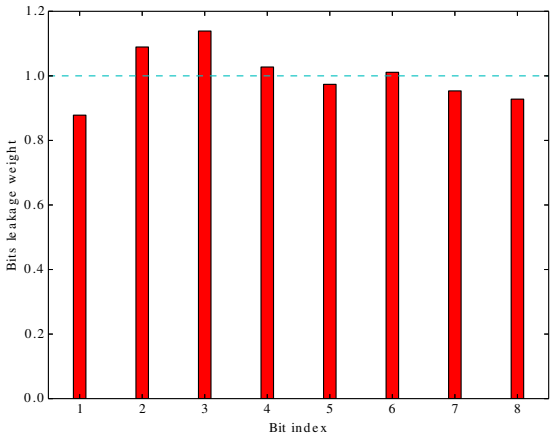


Fig. 5. Bit leakage weights obtained using linear regression.

H Example of an acquired EM trace

From Fig. 6, one can identify the presence of 3 spikes standing respectively to:

1. the loading of the Sbox input,
2. the Sbox access, and
3. the copy of the Sbox output.

The corresponding assembly code is detailed in Appendix I.

I Implementation details

We provide hereafter the instruction sequence used to perform an encoded PRESENT Sbox computation. The Sbox table is stored in register R0 and the Sbox input (*i.e.* the bitwise xor of the plaintext and the secret key) is stored in register R1.

```
1 EOR     R2,R2,R2    // clear R2
2 LDRB    R2,[R0,R1]  // access the Sbox and store the result in R2
3 EOR     R0,R0,R0    // clear R0
4 MOV     R0,R2       // copy the Sbox output into R0
```

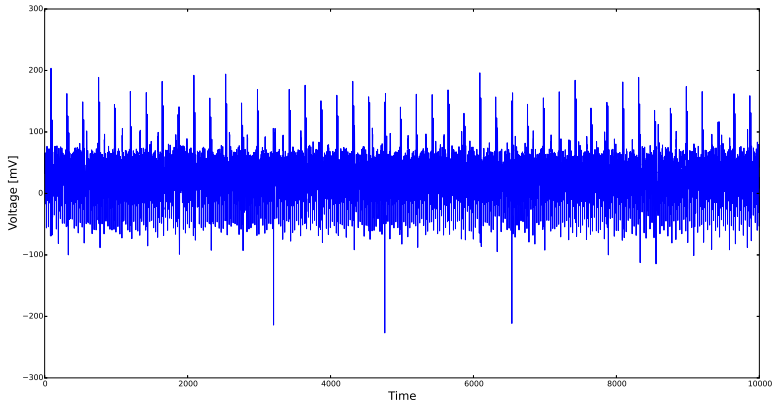


Fig. 6. Electromagnetic trace.

J Device characterization

Our goal here is to check if the STMF32 Micro-Controller registers have the same leakage model or leak differently. To do so, we have first implemented the following assembly code on copy #1. In fact, we performed two AES Sbox accesses and we stored the results into two different registers initialized at zero (Lines 2 and 5). The Sbox table is stored in register R0 and the Sbox input (*i.e.* the bitwise xor of the plaintext and the secret key) is stored in register R1.

```
1 EOR     R2,R2,R2    // clear R2
2 LDRB    R2,[R0,R1]  // access the AES Sbox and store the result in R2
3 MOV     R3,R2       // copy the AES Sbox output into R3
4 EOR     R3,R3,R3    // clear R3
5 LDRB    R3,[R0,R1]  // access the AES Sbox and store the result in R3
```

Then, we acquired 25.000 EM traces recording the two AES Sbox computations. Afterwards, we performed a linear regression attack to get the bit leakage weights of each registers. We plot the obtained results in Fig. 7.

From Fig. 7, two observations could be emphasized:

1. Both registers leaks closely to the Hamming weight model. This result is online with that shown in Appendix G, where the stochastic characterization was made for register R0.
2. Both registers R2 and R3 have roughly the same leakage model. Thus, to apply our scheme, a designer could use the same encoding function to protect all operations of a block cipher.

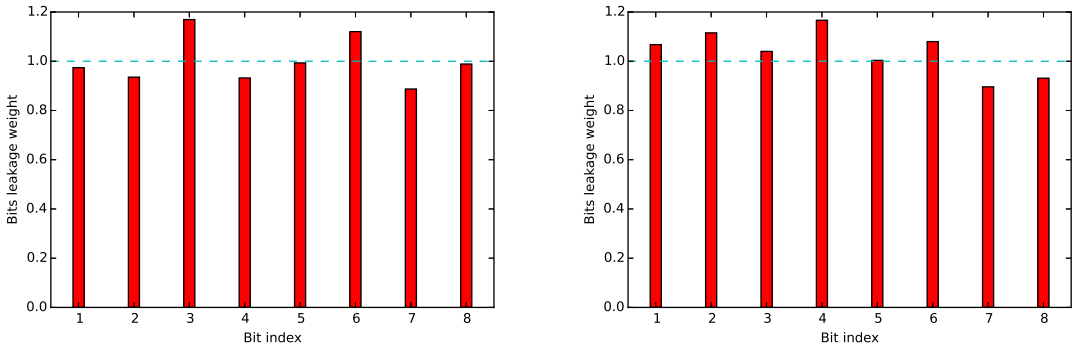


Fig. 7. Bit leakage weights obtained. Left: for register R2. Right: for register R3.

To confirm our claim, we applied an enhanced CPA attack (as described in Sec. 6.3) and estimated the evolution of the averaged rank of the correct key when targeting the Sbox output stored in $R2$ and $R3$. The result is plotted in Fig. 8. As expected, the key rank curves have the same convergence slope.

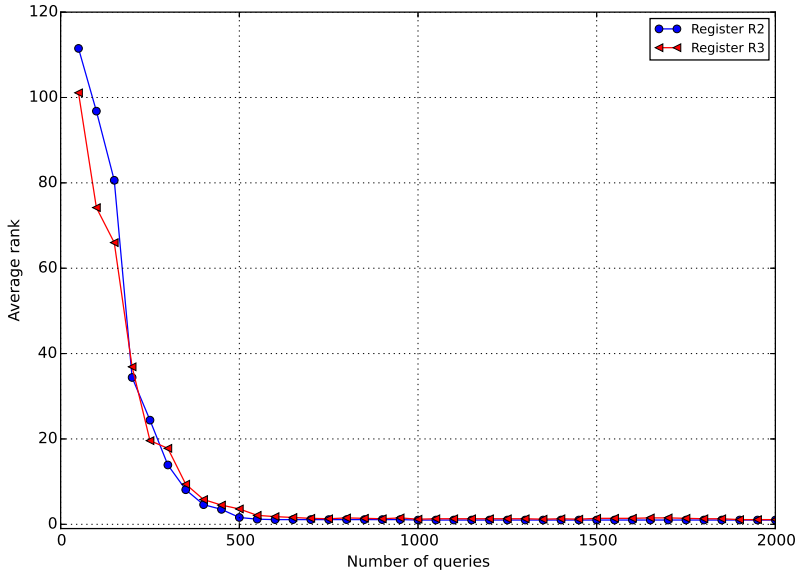


Fig. 8. Evolution of the correct key rank according to the number of observations.