Polynomial direct sum masking to protect against both SCA and FIA

Claude Carlet[*], Abderrahman Daif[*†], Sylvain Guilley[‡§], Cédric Tavernier[†]

November 27, 2017

Abstract

Side Channel Attacks (SCA) and Fault Injection Attacks (FIA) allow an opponent to have partial access to the internal behavior of the hardware. Since the end of the nineties, many works have shown that this type of attacks constitute a serious threat to cryptosystems implemented in embedded devices. In the state of the art, there exist several countermeasures to protect symmetric encryption (especially AES-128). Most of them protect only against one of these two attacks (SCA or FIA). A method called ODSM has been proposed to withstand SCA and FIA , but its implementation in the whole algorithm is a big open problem when no particular hardware protection is possible. In the present paper, we propose a practical masking scheme specifying ODSM which makes it possible to protect the symmetric encryption against these two attacks.

Keywords: Masking countermeasure, Error correcting codes, Side channel attack, Fault injection attack, AES.

1 Introduction

When an algorithm is implemented on a hardware device (Chip card, TPM, FPGA ...), the observable physical leakage (computing time, current consumption, electromagnetic radiation ...) can be exploited to mount so-called *side-channel attacks* (SCA). The most common countermeasures to combat such attacks are *masking* [3, 2] and *shuffling* [4]. *Shuffling* is a simple solution that involves randomizing a series of operations of the cipher so as to improve the SCA resistance. On the other hand, *masking* protects by mixing the sensitive data with some random value called the *mask*. The most generic known measure to protect against these attacks remains masking via homomorphic functions. However, it is still a challenging matter to build such function which at the same time is not intensive in terms of computation, so that it can be implemented on low-resource electrical components, and also passes all constitutive operations of a symmetric encryption, in particular the substitution-Box (also called S-Box, e.g. `SubBytes` for AES).

[*]LAGA, Department of Mathematics, University of Paris 8 (and Paris 13 and CNRS), Saint–Denis cedex 02, France.

[†]BU CSCS, Assystem E&OS, 23 Place de Wicklow, 78180 Montigny-le-Bretonneux, France

[‡]TELECOM-ParisTech, Crypto Group, Paris Cedex 13, France

[§]Secure-IC S.A.S. Rennes, France

Another mode of attack that threatens the electrical components, is called fault injection attacks (FIA). It consists in disrupting the operation of encryption or decryption by the injection of malicious faults into a cryptographic device and the observation of the corresponding erroneous outputs [14, 15]. Despite the high cost of equipment used in this type of attacks, it remains the most effective for obtaining information about the sensitive data. However, attacks vary depending on the type of cryptography targeted (symmetric or asymmetric) [16].

1.1 Related works

There exists several solutions that have been proposed to protect symmetric encryption from SCA. The most conventional masking method is to decompose the sensitive data x into several parts (shares) $x_0, x_1, \ldots, x_d$ such that $x = \bigoplus_{i=0}^{d} x_i$, then operate on each of the parts separately without involving the sensitive data in the calculation process. Each $x \mapsto F(x)$ transformation that composes the encryption (or decryption) algorithm must be replaced by a function $(x_0, \ldots, x_d) \mapsto (y_0, \ldots, y_d)$ such that $F(\sum_{i=0}^{d} x_i) = \sum_{i=0}^{d} y_i$ (which is called *masking* by abuse of language), and such that the knowledge of d shares manipulated when calculating this function gives no information about x (which is known as d-th order probing security). This method remains efficient and simple for linear transformations (XOR), however, it is still greedy in terms of calculations for the nonlinear part like `SubBytes` of AES. Since every function on a finite field is polynomial, it suffices to know how to mask the addition (XOR) and multiplication. The difficulty is to build shares $c_0, \ldots, c_d$ such that $\bigoplus_{i=0}^{d} c_i = ab$. Ishai *et al.* [12] implemented a solution which consists in securing the "NOT" and "AND" operations in a boolean circuit. If we consider two sensitive bits $b = \bigoplus_{i=0}^{d} b_i$ and $b' = \bigoplus_{i=0}^{d} b'_i$, we can compute $\neg b = \neg b_0 \oplus \bigoplus_{i=1}^{d} b_i$ ("$\neg$" denotes "NOT") and $bb' = \bigoplus_{i=0}^{d} \bigoplus_{j=0}^{d} b_i b'_j$. This solution makes it possible to obtain a d^{th}-order of security level [13], but the complexity in terms of computation and memory increases considerably according to the order (because the bitwise products $b_i b' j$ shall be masked before being summed up). Prouff and Rivan [13] proposed a generalization of this algorithm, in particular in $\mathbb{F}_{2^8}$. This solution, dedicated to the AES, allows to reach an order as high as required. However, the quadratic complexity of the calculations remains quite greedy for the components endowed with little resources. The order is therefore limited to the supported capacity of the component.

Another method is the so-called "Polynomial Masking", introduced separately by Prouff-Roche [11] and Goubin-Martinelli [9], which combines Shamir's Secret Sharing Scheme (SSS) [10] and secure multi-party computation techniques [17]. The masking operation of a sensitive data $m \in \mathbb{F}_{2^8}$, consists in constructing a function of degree d, such that $f_m(x) = m \oplus \bigoplus_{i=1}^{d} a_i x^i$, where $(a_i)_{1 \leq i \leq d}$ are some random secret coefficients, then as in the previous scheme m can be represented by d shares $(m_1, \ldots, m_d)$, with $m_i = (\alpha_i, f_m(\alpha_i))$ for $1 \leq i \leq d$ for some random inputs $(\alpha_i)_{1 \leq i \leq d}$. To get m (unmasked), we have to construct f_m (i.e calculate the coefficients $(a_i)_{1 \leq i \leq d}$) from $(\alpha_i, f_m(\alpha_i))_{1 \leq i \leq d}$ by polynomial interpolation, and finally calculate $m = f_m(0)$.

In [5] Bringer *et al.* are used LCD codes to construct the Orthogonal Direct Sum Masking (ODSM). This allows the sensitive data to be masked with a random mask, chosen uniformly from a set of codewords. In this scheme, a sensitive data $x \in \mathbb{F}_2^k$ is associated with a codeword in a vector subspace $\mathcal{C} \subsetneq \mathbb{F}_{2^n}$. The codeword is then XORed with a random value from the dual of $\mathcal{C}$ and we obtain the masked value: $z := xG + rH$, with $G \in \mathbb{F}_2^{k \times n}$ the generating matrix of $\mathcal{C}$, and $H \in \mathbb{F}_2^{k \times (n-k)}$ the parity-check matrix of $\mathcal{C}$, that is, the generating matrix of the dual of $\mathcal{C}$ (denoted $\mathcal{D}$). Moreover, the vector spaces $\mathcal{C}$ and $\mathcal{D}$ are supplementary, i.e. $\mathcal{C} \oplus \mathcal{D} = \mathbb{F}_2^n$, this means that $\forall z \in \mathbb{F}_2^n,\ \exists! (x, r) \in \mathcal{C} \times \mathcal{D} \mid z = xG + rH$. To recover the sensitive data x from z, it is sufficient to

calculate $zG^\top(GG^\top)^{-1}$. This scheme resists monovariate attacks of degree $d_\mathcal{C} - 1$ ($d_\mathcal{C}$ denotes the minimal distance of $\mathcal{C}$) without increasing considerably the memory space used. Also, to the best of our knowledge, it is the only masking schemes that combines both masking and error detection. It allows to detect errors with a probability $1 - 2^{-n+k}$, assuming the attacker is able to inject faults uniformly in $\mathbb{F}_2^n$.

In [1], Azzi *et al.* presented a countermeasure against fault injection attacks. This method consists in encoding the sensitive data x using a systematic linear code. Let us consider $G = (I|A)$ the generating matrix of a linear code $\mathcal{C}$ in a systematic form (I denotes the identity matrix), $\texttt{Encode}(x) := xG = (x|xA)$ the encoding operation, and f a non linear transformation of AES (`SubBytes` for example). Before starting the encryption process, three tables $T0 : x \mapsto xA$, $T1 : x \mapsto f(x)$ and $T2 : xA \mapsto f(x)A$ must be prefilled. Thus, using these three tables, we can compute $f((x|xA)) = (f(x)|f(x)A)$, and thanks to the added redundancy $(f(x)A)$ we can detect the error injections according to the capacity of the chosen code. This method makes it possible to detect errors; in addition, it is possible to combine it with existing masking methods by applying a mask to the three tables (i.e instead of using x, we can use $x + r$). The masked version of this method is a special case of the DSM family, in which the sensitive data x and the random mask r are encoded using the same code ($z = xG + rG = (x + r)G$). The advantage compared to the previous construction (ODSM) is that this scheme makes it easier to decode, since the masked word is already a codeword. On the other hand, the disadvantage is that the mask remains identical throughout the encryption process because the tables $T0, T1, T2$ depend on it.

As shown in the figure below, the DSM (ODSM without orthogonality) is a generalization of ODSM and of Inner Product Masking (IPM) [8], which in turn is a generalization of traditional masking.

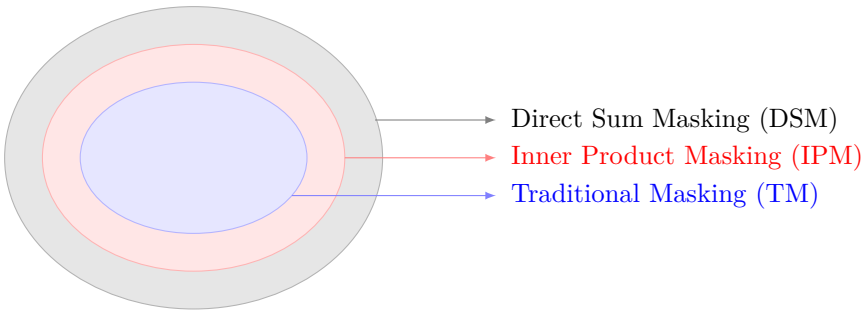

Figure 1: Comparison between different masking schemes

1.2 Our contribution

Our contribution is based on the last proposed solution DSM: it consists in designing a software masking scheme of AES transformations, able to detect and correct errors that can be injected, and furthermore, minimize the costs in terms of memory and computing time as well.

Let s and s' be two sensitive data, we denote by $\texttt{Mask}(s)$ the masked word of s, the challenge is to design two homomorphic functions `Add` and `SMult`, such that: $\texttt{Add}(\texttt{Mask}(s), \texttt{Mask}(s')) = \texttt{Mask}(s+s')$ and $\texttt{SMult}(\texttt{Mask}(s), \texttt{Mask}(s')) = \texttt{Mask}(ss')$. With these two operations we can rebuild the masked version of AES, and redefine each of its transformations (namely: `XOR, MixColumns, SubBytes`). In addition, to be able to detect and correct injected errors, the output space of the masking

operation must be an error correcting code. To the best of our knowledge, according to the state of the art, there is not yet such a masking which combines these two characteristics (software masking and detect/correct errors).

In [5] Bringer *et al.* are used LCD codes, as recalled above. This allows the sensitive data to be masked with a random mask, chosen uniformly from a set of codewords. In addition, the unmasking does not need to store the chosen mask (except in the case where one wants to detect the errors). The approach that we will present in this paper is somewhat similar. We have chosen to work on what we shall call a polynomial field, that is $\mathbb{F}_2[x]/p(x)$ where $p(x)$ is an irreducible polynomial. The operations in this field are commutative, which is not the case in [5], since matrix products are *not* commutative in general; this gives us more flexibility to build the homomorphic function.

In our scheme, the same operation allows to mask and encode the sensitive information, as in [1], but this operation does not need to store the table of all possible inputs of the S-box as in the previous citation.

2 Preliminaries

Let $K = (\mathbb{F}_2[x]/p(x), +, \cdot)$ be a polynomial field modulo an irreducible polynomial $p(x)$ of degree k (we can take in particular $p(x) = x^8 + x^4 + x^3 + x + 1$ for `SubBytes` and `MixColumns` transformations of the AES [6]). Each equivalence class of this field is represented by a polynomial of degree at most $k-1$, (for $k = 8$ the polynomial can be represented by two hexadecimal digits). We denote by K^n an n-dimensional vector space over K.

Let $\mathcal{C}$ and $\mathcal{D}$ be two 1-dimensional vector sub-spaces of K^n, whose elements will be denoted with arrows for helping to distinct between scalars and vetors. Let $\vec{g},\ \vec{h} \in K^n \setminus \{\vec{0}\}$ be generators of $\mathcal{C}$ and $\mathcal{D}$ respectively such that $\langle \vec{g}, \vec{h} \rangle = 0$, $\langle \vec{g}, \vec{g} \rangle \neq 0$, and $\langle \vec{h}, \vec{h} \rangle \neq 0$. So we have:

$$\mathcal{C} = \left\{ s \cdot \vec{g},\ \forall s \in K \right\} ,$$

and

$$\mathcal{D} = \left\{ s \cdot \vec{h},\ \forall s \in K \right\} ,$$

where " $\cdot$ " denotes the product between a scalar and a vector (i.e. $s \cdot \vec{g} = (sg_1, \ldots, sg_n) \in K^n$); this operation is also known as a scaling of vector $\vec{g}$ by scalar s. We denote $\mathcal{C} + \mathcal{D} \subset K^n$ by $\mathcal{K}$, and we have: $\forall \vec{z} \in \mathcal{K}, \exists! (\vec{c}, \vec{d}) \in \mathcal{C} \times \mathcal{D}$ such that $\vec{z} = \vec{c} + \vec{d}$. For $n = 2$, $\mathcal{C}$ is an LCD code [18].

3 The masking operations

To proceed with the masking, we need to construct a homomorphic function for each of the operations that compose the symmetric cryptosystem, in particular the addition modulo 2 (XOR) and the multiplication over K. To mask a sensitive byte $s \in K$, first we calculate the corresponding codeword in $\mathcal{C}$, then we add a random word of the dual $\mathcal{D}$ which will act as the mask:

$$\texttt{Mask}(s) := s \cdot \vec{g} + r \cdot \vec{h} , \tag{1}$$

for some random value $r \in K$.

Algorithm 1 $\texttt{Mask}(s)$ — complexity $O(n)$

Input: a sensitive data $s \in K$
Output: $\texttt{Mask}(s) \in \mathcal{K}$
$r \leftarrow K$ ▷ a random value
$\vec{z} \leftarrow 0 \in K^n$
for $1 \leq i \leq n$ **do**
    $z_i \leftarrow s g_i + r h_i$
end for
return $\vec{z}$

If $\vec{z} = \texttt{Mask}(s)$, then to extract the sensitive data s hidden in $\vec{z}$, we calculate:

$$\texttt{unmask}(\vec{z}) := \langle \vec{z}, \vec{g} \rangle \langle \vec{g}, \vec{g} \rangle^{-1} \ , \tag{2}$$

where "$\langle \cdot, \cdot \rangle$" denotes the usual inner product.
Correctness. We have:

$$\begin{aligned} \langle \vec{z}, \vec{g} \rangle \langle \vec{g}, \vec{g} \rangle^{-1} &= \langle s \cdot \vec{g} + r \cdot \vec{h}, \vec{g} \rangle \langle \vec{g}, \vec{g} \rangle^{-1} \\ &= \big(s \langle \vec{g}, \vec{g} \rangle + \underbrace{r \langle \vec{h}, \vec{g} \rangle}_{=0} \big) \langle \vec{g}, \vec{g} \rangle^{-1} \\ &= s \ . \end{aligned}$$

Algorithm 2 $\texttt{Unmask}(\vec{z})$ — complexity $O(n)$

Input: a masked value $\vec{z} = \texttt{Mask}(s) \in \mathcal{K}$
Output: $s \in K$
$s \leftarrow 0 \in K$
for $1 \leq i \leq n$ **do**
    $s \leftarrow s + z_i g_i$
end for
$s \leftarrow s \|g\|^{-2}$ ▷ We suppose that $\|g\|^{-2} = \langle \vec{g}, \vec{g} \rangle^{-1}$ is precalculated
return s

This masking operation is a linear function, it is therefore obvious that the `AddRoundKey` transformation remains unchangeable, if we consider $\texttt{Mask}(s)$ and $\texttt{Mask}(k)$ the masked value of the cipher and the round key respectively, then the masked value of $s + k$ can be calculated as follows:

$$\texttt{Mask}(s + k) = \texttt{Mask}(s) + \texttt{Mask}(k) \ . \tag{3}$$

Algorithm 3 $\texttt{Add}(\vec{z}, \vec{z}')$ — complexity $O(n)$

Input: two masked values: $\vec{z} = \texttt{Mask}(s), \vec{z}' = \texttt{Mask}(s') \in \mathcal{K}$
Output: $\texttt{Mask}(s + s') \in \mathcal{K}$
$\vec{y} \leftarrow 0 \in \mathcal{K}$
for $1 \leq i \leq n$ **do**
    $y_i \leftarrow z_i + z'_i$
end for
return $\vec{y}$

However, for `MixColumns` and `SubBytes` transformations which are composed of polynomial products over K, two types of operations can be distinguished:

- The product between a sensitive data and a non-sensitive data, as is the case in the `MixColumns` transformation, since its coefficients are public, the same case for `SubBytes` coefficients. For this type of operations there is no need to mask the public coefficients. Thus, to mask an operation λs for some public coefficient λ and sensitive data s we proceed thereby:

$$\texttt{Mask}(\lambda s) := \lambda \cdot \texttt{Mask}(s) = (\lambda z_1, \ldots, \lambda z_n) \ . \tag{4}$$

Algorithm 4 $\texttt{Mult}(\lambda, \vec{z})$ complexity $O(n)$

1: **Input:** A public data $\lambda \in K$ and masked values $\vec{z} = \texttt{Mask}(s) \in \mathcal{K}$
2: **Output:** $\texttt{Mask}(\lambda s) \in \mathcal{K}$
3: $\vec{y} \leftarrow 0 \in \mathcal{K}$
4: **for** $1 \leq i \leq n$ **do**
5: $\quad y_i \leftarrow \lambda z_i$
6: **end for**
7: **return** $\vec{y}$

- The product between two sensitive data:

$$\texttt{Mask}(ss') := \langle \vec{g}, \vec{g} \rangle^{-1} \cdot \Big(\texttt{Mask}\Big(r_t + \langle \texttt{Mask}(s), \texttt{Mask}(s') \rangle \Big) - \Big[r_t + \langle \vec{h}, \vec{h} \rangle^{-1} \langle \texttt{Mask}(s), \vec{h} \rangle \langle \texttt{Mask}(s'), \vec{h} \rangle \Big] \cdot \vec{g} \Big) \ . \tag{5}$$

correctness. we have:

$$\begin{aligned} r_t + \langle \texttt{Mask}(s), \texttt{Mask}(s') \rangle &= r_t + \langle s \cdot \vec{g} + r \cdot \vec{h}, s' \cdot \vec{g} + r' \cdot \vec{h} \rangle \\ &= r_t + \langle s \cdot \vec{g}, s' \cdot \vec{g} \rangle + \langle s \cdot \vec{g}, r' \cdot \vec{h} \rangle + \langle r \cdot \vec{h}, s' \cdot \vec{g} \rangle + \langle r \cdot \vec{h}, r' \cdot \vec{h} \rangle \\ &= r_t + ss' \langle \vec{g}, \vec{g} \rangle + \underbrace{sr' \langle \vec{g}, \vec{h} \rangle + s' r \langle \vec{h}, \vec{g} \rangle}_{=0} + rr' \langle \vec{h}, \vec{h} \rangle \\ &= r_t + ss' \langle \vec{g}, \vec{g} \rangle + rr' \langle \vec{h}, \vec{h} \rangle \ . \end{aligned}$$

$$\begin{aligned} \langle \texttt{Mask}(s), \vec{h} \rangle &= \langle s \cdot \vec{g} + r \cdot \vec{h}, \vec{h} \rangle \\ &= s \langle \vec{g}, \vec{h} \rangle + r \langle \vec{h}, \vec{h} \rangle \\ &= r \langle \vec{h}, \vec{h} \rangle \ . \end{aligned}$$

So we get:

$$\begin{aligned} &\langle \vec{g}, \vec{g} \rangle^{-1} \cdot \Big(\texttt{Mask}\Big(r_t + \langle \texttt{Mask}(s), \texttt{Mask}(s') \rangle \Big) - \Big[r_t + \langle \vec{h}, \vec{h} \rangle^{-1} \langle \texttt{Mask}(s), \vec{h} \rangle \langle \texttt{Mask}(s'), \vec{h} \rangle \Big] \cdot \vec{g} \Big) \\ &\quad = \langle \vec{g}, \vec{g} \rangle^{-1} \cdot \Big(\Big(r_t + ss' \langle \vec{g}, \vec{g} \rangle + rr' \langle \vec{h}, \vec{h} \rangle \Big) \cdot \vec{g} + r'' \cdot \vec{h} - \Big(r_t + rr' \langle \vec{h}, \vec{h} \rangle \Big) \cdot \vec{g} \Big) \\ &\quad = \langle \vec{g}, \vec{g} \rangle^{-1} \cdot \Big(ss' \langle \vec{g}, \vec{g} \rangle \cdot \vec{g} + r'' \cdot \vec{h} \Big) \\ &\quad = ss' \cdot \vec{g} + r'' \langle \vec{g}, \vec{g} \rangle^{-1} \cdot \vec{h} \\ &\quad = \texttt{Mask}(ss') \ , \end{aligned}$$

where r'' is a fresh random value.

Because the product rr' is not uniformly distributed in K, we have introducted a temporary mask r_t to secure the inner product: $\langle \texttt{Mask}(s), \texttt{Mask}(s') \rangle$.

Algorithm 5 $\texttt{SMult}(\vec{z}, \vec{z}')$ complexity $O(5n)$

Input: Two masked values $\vec{z} = \texttt{Mask}(s), \vec{z}' = \texttt{Mask}(s') \in \mathcal{K}$
Output: $\texttt{Mask}(ss') \in \mathcal{K}$
$r_t \leftarrow K^*$ ▷ A temporary mask
$a \leftarrow r_t$
$b \leftarrow 0 \in K$
$c \leftarrow 0 \in K$
$\vec{y} \leftarrow 0 \in \mathcal{K}$
for $1 \leq i \leq n$ **do**
    $a \leftarrow a + z_i z'_i$
    $b \leftarrow b + z_i h_i$
    $c \leftarrow c + z'_i h_i$
end for
$\vec{y} \leftarrow \|g\|^{-2} \left(\texttt{Mask}(a) - \texttt{Mult}(r_t + bc\|h\|^{-2}, \vec{g}) \right)$ ▷ We suppose that $\|g\|^{-2}$ and $\|h\|^{-2}$ are precalculated ($\|g\|^{-2} = \langle \vec{g}, \vec{g} \rangle^{-1}$, $\|h\|^{-2} = \langle \vec{h}, \vec{h} \rangle^{-1}$)
return $\vec{y}$

4 Security and performance

Property 1. *Let $\vec{g}$ and $\vec{h}$ be two vectors in K^n such that $\langle \vec{g}, \vec{h} \rangle = 0$, $\mathcal{C} = \{s \cdot \vec{g} \mid \forall s \in K\}$, and $\mathcal{D} = \{s \cdot \vec{h} \mid \forall s \in K\}$. The condition "$\langle \vec{g}, \vec{g} \rangle \neq 0$ and $\langle \vec{h}, \vec{h} \rangle \neq 0$" implies:*

1. *$\vec{g}$ and $\vec{h}$ are linearly independent;*
2. *$\mathcal{C} \cap \mathcal{D} = \{0\}$;*
3. *$\forall \vec{z} \in \mathcal{K},\ \exists!\ (s, r) \in K^2$ such that $\vec{z} = s \cdot \vec{g} + r \cdot \vec{h}$.*

Proof. The two first properties are obvious and the third is a direct consequence of the rank-nullity theorem, $dimension(\mathcal{K}) = dimension(\mathcal{C}) + dimension(\mathcal{D}) = 2$. □

Property 2. *Let $(\mathbb{F}_2[x]/p(x), +)$ be an additive group, with $p(x)$ a polynomial of degree k, and $\mathbb{F}_2^k$ a vector space of dimension k. There exists a morphism δ defined by:*

$$\begin{array}{rccc} \delta & : & \mathbb{F}_2[x]/p(x) & \rightarrow & \mathbb{F}_2^k \\ & & \sum_{i=0}^{k-1} c_i x^i & \mapsto & (c_0, \ldots, c_{k-1}) \end{array},$$

such that: $\forall u, u' \in \mathbb{F}_2[x]/p(x)$, we have: $\delta(u + u') = \delta(u) + \delta(u') \in \mathbb{F}_2^k$.

Let us denote by $\delta^n : (\mathbb{F}_2[x]/p(x))^n \rightarrow \mathbb{F}_2^{kn}$ defined by :

$$\delta_n(u_1, \ldots, u_n) = (\delta(u_1), \ldots, \delta(u_n)) \ .$$

This passage from a finite field K to a vector space will be useful for the decoding part, since we will decode the binary representation of the codewords.

The product $s \cdot \vec{g}$ between an element $s \in K$ and a vector $\vec{g} \in K^n$ corresponds to a product between a vector $\delta(s) \in \mathbb{F}_2^k$ and a matrix $G \in \mathbb{F}_2^{k \times kn}$, such that the i-th arrow of G corresponds to the binary representation of $x^i \cdot \vec{g}$ (i.e. $G_i = \delta_n(x^i\vec{g}_1, \ldots, x^i\vec{g}_n)$.

Example 1. *Let $s = 1 + x + x^5 \in K$, and $\vec{g} = (1, 1, x^3 + x^4 + x^7) \in K^3$ with $K = \mathbb{F}_2[x]/x^8 + x^4 + x^3 + x + 1$, we have: $\delta(s) = (11000100)$ and*

$$G = \begin{array}{cccccccccccccccccccccccc} \overbrace{}^{\delta(x^i g_0)} & & & & & & & & \overbrace{}^{\delta(x^i g_1)} & & & & & & & & \overbrace{}^{\delta(x^i g_2)} & & & & & & & & \\ 1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1 & \delta_n(\vec{g}) \\ 0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&1&0&1&0&0 & \delta_n(x \cdot \vec{g}) \\ 0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&1&0&1&0 & \delta_n(x^2 \cdot \vec{g}) \\ 0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&1&0&1 & \delta_n(x^3 \cdot \vec{g}) \\ 0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&1&0 & \delta_n(x^4 \cdot \vec{g}) \\ 0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&1 & \delta_n(x^5 \cdot \vec{g}) \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&1&0&1&0&0&0 & \delta_n(x^6 \cdot \vec{g}) \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&1&0&1&0&0 & \delta_n(x^7 \cdot \vec{g}) \end{array}$$

thus :

$$\begin{aligned} \delta_n(s \cdot \vec{g}) &= (\delta(x^5 + x + 1), \delta(x^5 + x + 1), \delta(x^5 + x^4 + x^2 + 1)) \\ &= (110001001100010010101100) \\ &= \delta(s).G \end{aligned}$$

Definition 1 (Generating matrix)**.** *Let $\mathcal{C} = \{s \cdot \vec{g} \mid \forall s \in K\}$ be a vector subspace of K^n, with $\vec{g} = (g_1, \ldots, g_n)$, the generating matrix of $\mathcal{C}$ can be defined as: $G \in \mathbb{F}_2^{k \times nk}$ such that $G_i = \delta_n(x^i \cdot \vec{g})$ for $1 \leq i \leq k$.*

Let us denote by $\mathcal{C}' = \{vG \mid \forall v \in \mathbb{F}_2^k\}$ the linear code that corresponds to $\mathcal{C}$ with parameters $[nk, k, d_{\mathcal{C}'}]$, with $d_{\mathcal{C}'}$ the minimal distance of $\mathcal{C}'$.

Theorem 1. *The masking algorithm (Algo. 1) is secure against a monovariate SCA at order $j < d_{\mathcal{C}}$.*

Proof. As in the foregoing, let $\vec{g}, \vec{h} \in K^n$ be two generators of $\mathcal{C}$ and $\mathcal{D}$ respectively, such that: $\langle \vec{g}, \vec{h} \rangle = 0$, $\langle \vec{g}, \vec{g} \rangle \neq 0$ and $\langle \vec{h}, \vec{h} \rangle \neq 0$. The masking operation which consists in calculating $\vec{z} = s \cdot \vec{g} + r \cdot \vec{h}$ for a sensitive data s with a random value r, is equivalent to $z' = sG + rH$, with G and H two generating matrices of $\mathcal{C}$ and $\mathcal{D}$ respectively. It is thus observed that this masking operation forms part of the DSM family. The Theorem 2 in [5], proves that the last masking operation ($z' = sG + rH$) can be attacked by monovariate high-order SCA only at order $j \geq d_{\mathcal{C}}$. The only difference between the construction that we are presenting in this article and the one in the article cited previously, is the orthogonality. Indeed, in ODSM, the matrices G and H are supposed to be orthogonal (i.e. $GH^\top = 0$), which is not the case for us. However, orthogonality is not required for the theorem, because, if we consider $\Phi : \mathbb{F}_2^{nk} \to \mathbb{R}$ a leakage function of numerical degree $j < d_{\mathcal{C}}$, the Proposition 3 in [5] demonstrates that there is no linear dependency between the leakage $\Phi(\texttt{Mask}(s))$ and the sensitive data s. □

4.1 Algorithmic complexity

Table 1 summarizes the algorithmic complexity of each of the algorithms we have presented, this complexity is calculated with respect to the size of the mask, and the number of operations (addition and multiplication) performed.

	nb. multiplication	nb. XORs
`Mask`	$2n$	n
`Unmask`	$n+1$	n
`Add`	0	n
`Mult`	n	0
`SMult`	$6n+3$	$4n+2$

Table 1: An overall view of the complexity of each function

The algorithmic complexity of the masking depends on the complexity of the multiplication algorithm (`SMult`). This complexity is expressed as a function of the length of the masked word n. In this article, the length n depends on the minimum distance of the code $\mathcal{C}$, contrary in the state of the art (except ODSM) n depends directly on the order of the masking ($n = 2d+1$). In table 2 we present a comparison between our method and other state of the art schemes in terms of complexity.

	Order d	Masked word length n	Secure multiplication complexity
This paper	1	3	$6n+3=21$
	2	3	$6n+3=21$
	3	3	$6n+3=21$
	4	3	$6n+3=21$
[11]	1	3	$4d^3+8d^2+3d=15$
	2	5	$4d^3+8d^2+3d=70$
	3	7	$4d^3+8d^2+3d=189$
	4	9	$4d^3+8d^2+3d=396$
[8]	1	3	$2n^2-n=15$
	2	5	$2n^2-n=45$
	3	7	$2n^2-n=91$
	4	9	$2n^2-n=153$

Table 2: Secure multiplication complexity, comparison with other state of the art schemes

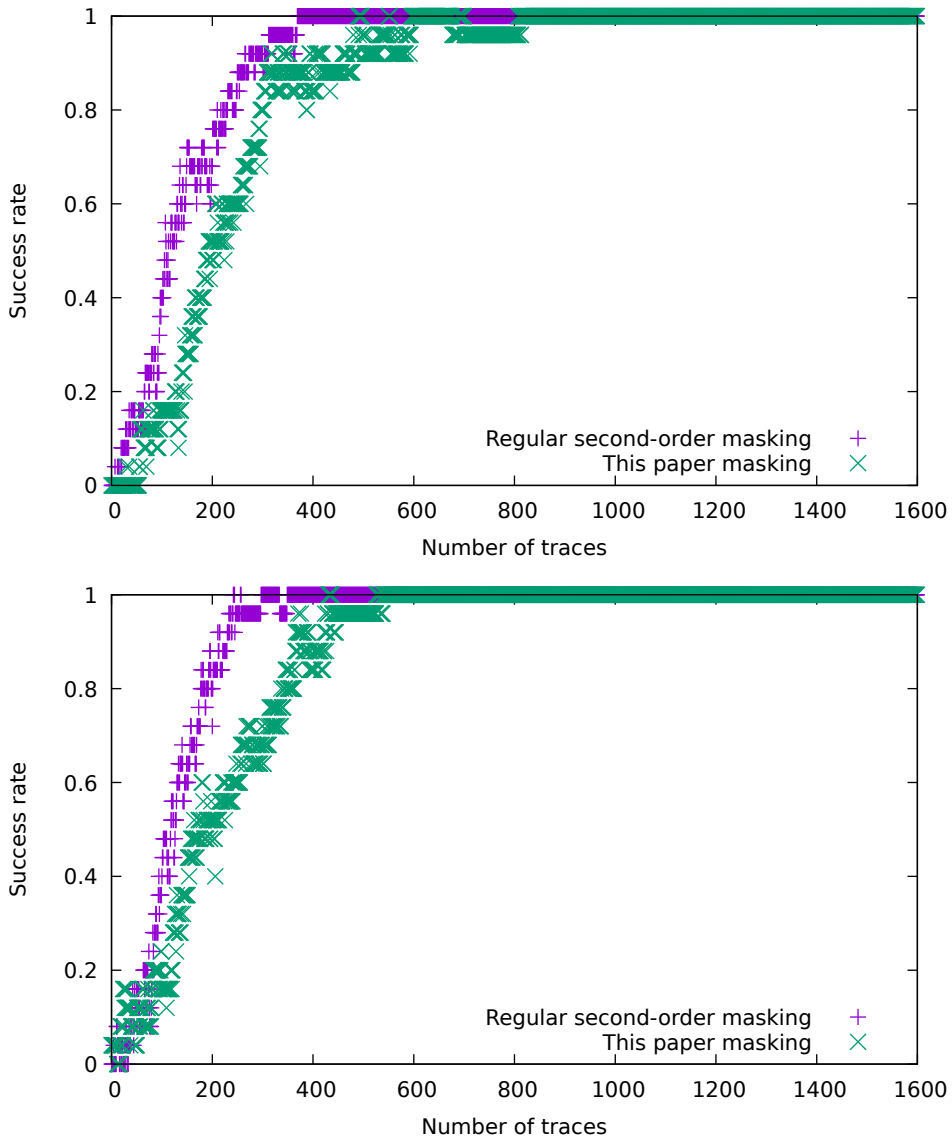

Figure 2: A comparison of the success rate of the attack in theorem 1 in [19] compared to the number of traces between the regular masking of second order $(s \oplus r|r)$ and the proposed masking (with $n = 3$).

4.2 Example (S-Box)

The `SubBytes` transformation of AES is mathematically defined by [7]:

$$\texttt{SubBytes}(s) = 63 + 05s^{-1} + 09s^{-2} + F9s^{-4} + 25s^{-8} + F4s^{-16} + 01s^{-32} + B5s^{-64} + 8Fs^{-128} \in K \ ,$$

the coefficients are polynomials in K, and are represented by hexadecimal bytes. This transformation encompasses the three types of operation we have seen previously:

- The addition (XOR),
- The multiplication between a public data (the coefficients) and a sensitive data (s),
- The multiplication between the sensitive data and itself (the powers of s).

Before defining the masked version of `SubBytes`, we need to define the function `Inverse`.

Definition 2. *The Euler totient function is defined, for integer $n \geq 1$, by :*

$$\varphi(n) := \mathbf{card}\{x \leq n, \mid gcd(n, x) = 1\} \ ,$$

i.e. the number of integers in the range $[1, n]$ that are relatively prime to n.

Theorem 2. *In number theory, Euler's theorem states that if n and a are coprime positive integers, then :*

$$a^{\varphi(n)} \equiv 1 \pmod{n} \ .$$

For the AES case, $K = \mathbb{F}_2[x]/x^8 + x^4 + x^3 + x + 1 \approx \mathbb{F}_{2^8}$. So we have $\forall s \in K, s^{\varphi(p(x))} = 1$ with φ the Euler totient function, and $p(x) = x^8 + x^4 + x^3 + x + 1$. We know that p is an irreducible polynomial, which implies that $\forall a \in \mathbb{F}_{2^8} \setminus \{0\}$, a does not devide p, and hence $\varphi(p(x)) = 255$, we conclude that: $s^{-1} = s^{\varphi(p(x))-1} = s^{254} \in K$. Bruneau N., Guilley S., Heuser A., Rioul O., Standaert FX., Teglia Y. (2016) Taylor Expansion of Maximum Likelihood Attacks for Masked and Shuffled Implementations. In: Cheon J., Takagi T. (eds) Advances in Cryptology – ASIACRYPT 2016. ASIACRYPT 2016. Lecture Notes in Computer Science, vol 10031. Springer, Berlin, Heidelberg

To calculate s^{254}, we are going to use the "Left-to-right binary algorithm", it consists in scanning the bits of the exponent in left to right order. If the bit is 1, the algorithm calculates the square of the previous result (Which is initialized to 1) multiplied by s, otherwise it calculates just the square. In our case the exponent is $254 = 11111110_2$. So the first 7 operations are square and multiplication, and the 8-th operation (associated with bit 0) is just a square. We can also see that : $s^{254} = s^{2(1+2(1+2(1+2(1+2(1+2\times1))))))} = \left(\left(\left(\left(\left(\left(s^{2\times1}s\right)^2s\right)^2s\right)^2s\right)^2s\right)^2s\right)^2$.

Algorithm 6 `Inverse`$(\vec{z})$ — complexity $O(65n)$

Input: a masked value $\vec{z} = \texttt{Mask}(s) \in \mathcal{K}$
Output: $\texttt{Mask}(s^{-1}) \in \mathcal{K}$
$\vec{y} \leftarrow \vec{z}$ ▷ $\vec{y} = \texttt{Mask}(1^2 s)$
for $1 \leq i \leq 6$ **do**
    $\vec{y} \leftarrow \texttt{SMult}(\texttt{SMult}(\vec{y}, \vec{y}), \vec{z})$
end for
$\vec{y} \leftarrow \texttt{SMult}(\vec{y}, \vec{y})$
return $\vec{y}$

Algorithm 7 `MSubBytes`$(\vec{z})$ complexity $O(117n)$

Input: a masked values $\vec{z} = \texttt{Mask}(s) \in \mathcal{K}$
Output: $\texttt{Mask}(\texttt{SubBytes}(s)) \in \mathcal{K}$
$\vec{y} \leftarrow \texttt{Inverse}(\vec{z})$ ▷ $\vec{y} = \vec{z}^{-1}$
$\vec{u} \leftarrow \texttt{Mask}(63)$
$\lambda = \{05, 09, F9, 25, F4, 01, B5\}$ ▷ The `SubBytes` coefficients
for $1 \leq i \leq 7$ **do**
    $\vec{u} \leftarrow \texttt{Add}(\vec{u}, \texttt{Mult}(\lambda_i, \vec{y}))$ ▷ $\vec{u} = \vec{u} + \lambda \cdot \vec{y}$
    $\vec{y} \leftarrow \texttt{SMult}(\vec{y}, \vec{y})$
end for
$\vec{u} \leftarrow \texttt{Add}(\vec{u}, \texttt{Mult}(8F, \vec{y}))$ ▷ $\vec{u} = \vec{u} + 8F \cdot \vec{y}$
return $\vec{u}$

	nb. multiplication	nb. XORs
`Inverse`	$78n + 39$	$52n + 26$
`MSubBytes`	$130n + 60$	$89n + 40$

Table 3: The complexity of masked version of `SubBytes` transformation

5 Detection and correction of errors

In order to detect and correct errors, it is important that the masked value be a codeword, for $n = 2$ the output space of $\vec{z} = s \cdot \vec{g} + r \cdot \vec{h}$ is the vector space $\mathbb{F}_2^{16}$, the minimum distance being 1, so it is impossible to detect errors. To sweep this, it is necessary that the output space be an error correcting code with a minimum distance greater than 2. We therefore propose to increase the size of the starting space to $n = 3$, and thus we work on $\mathcal{K} = K^3$ instead of K^2.

Let $\vec{g}, \vec{h} \in K^3 \setminus \{\vec{0}\}$ be two generators so that $\langle \vec{g}, \vec{h} \rangle = 0$, $\mathcal{C} = \{s \cdot \vec{g} = (sg_1, sg_2, sg_3) \mid \forall s \in K\}$ the linear code generated by $\vec{g}$, and $\mathcal{D} = \{r \cdot \vec{h} = (rh_1, rh_2, rh_3) \mid \forall r \in K\}$ its dual. The cardinal of $\mathcal{C}$ and $\mathcal{D}$ is $|\mathcal{C}| = |\mathcal{D}| = 2^8$. Let $G \in \mathbb{F}_2^{8\times 24}$ and $H \in \mathbb{F}_2^{8\times 24}$ be the generating matrix of $\mathcal{C}$ and $\mathcal{D}$ respectively. We note that the two matrices G and H are not orthogonal ($G.H^\top \neq 0$). Let $\mathcal{K} = \mathcal{C} \oplus \mathcal{D}$ be the set of masked words, i.e. $\mathcal{K} = \left\{\vec{c} + \vec{d} \mid \forall (\vec{c}, \vec{d}) \in \mathcal{C} \times \mathcal{D}\right\}$. The generating matrix of $\mathcal{K}$ is $J = \left(\frac{G}{H}\right) \in \mathbb{F}_2^{16\times 24}$.

Property 3. *$\mathcal{K}$ is a linear code, with the correction capacity equal $\left\lceil \frac{d_\mathcal{K}}{2} - 1 \right\rceil$, such that $d_\mathcal{K}$ denotes the minimum distance of $\mathcal{K}$.*

Property 4. *Let us denote L the parity matrix of $\mathcal{K}$, i.e. $JL^\top = 0$. Let $z = sG + rH = (s, r)J$ a masked value, we have: $zL^\top = 0$.*

Property 5 (Single error correction)**.** *If we consider $z' = z + e$ the erroneous masked word, with e the error vector. The syndrome decoding consists in calculating $\varepsilon = z'L^\top = eL^\top$. If the syndrome is not zero, which means an error injection, then the error position corresponds to the position of the column of L that equal ε.*

Example. In this example, for $n = 3$, and $\vec{g} = (1, 1, x^7 + x^4 + x^3), \vec{h} = (x^7 + x^4 + x^3 + 1, 1, 1)$ the minimal distance of $\mathcal{K}$ is $d_{\mathcal{K}} = 3$, thus the code can correct at most one error, which is sufficient to protect against single fault injection attack.

$$J = \left[\begin{array}{cccccccc|cccccccc|cccccccc}
1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1\\
0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&1&0&1&0&0\\
0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&1&0&1&0\\
0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&0&1&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&1&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&1\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&1&0&1&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&1&1&1&0&1&0&0\\
\hline
1&0&0&1&1&0&0&1&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&1&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&1&0&0&1&0&1&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\
1&1&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0&0\\
0&1&1&0&0&1&0&1&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0&0\\
1&1&1&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&0\\
0&1&1&1&0&1&0&1&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1
\end{array}\right],$$

We denote by L the parity matrix of $\mathcal{K}$:

$$L = \left[\begin{array}{cccccccc|cccccccc|cccccccc}
1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0&1&0&1&0\\
0&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&1&1&1&1\\
0&0&1&0&0&0&0&0&1&0&1&1&0&0&0&0&1&0&1&0&0&1&1&1\\
0&0&0&1&0&0&0&0&1&0&0&1&1&0&0&0&0&1&0&1&1&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0&1&0&0&1&1&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0&1&0&0&1&1\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&0&0&1&0&1&0&0&1\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0&1&0&1&0&0
\end{array}\right].$$

Let $s = x^7 + x^3 + 1$ be a sensitive data, and $\vec{z} = \texttt{Mask}(s) = (x^5 + x + 1, x^6 + x^4 + x^3 + 1, x^7 + x^6 + x^3 + x)$, imagine that $\vec{z}$ has undergone an error at a position $i = 13$ which corresponds to a vector $\vec{e} = (0, x^4, 0)$, we thus obtain, $\vec{z}' = (x^5 + x + 1, x^6 + x^3 + 1, x^7 + x^6 + x^3 + x) \in \mathcal{K}$ that corresponds to $y = (1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1) \in \mathbb{F}_2^{24}$. The syndrome $\varepsilon = y.L^\top = (0, 0, 0, 1, 1, 0, 0, 0)$ corresponds to the 13th column of the matrix L which is the error position.

In this example, the minimum distance of $\mathcal{C}'$ and $\mathcal{D}'$ is 5, thus, the masking operation is secure against a monovariate SCA at order at most 4. The detection of error in this case ($n = 3$) costs 24 XORs, and the correction costs 25 XORs.

6 Conclusion

In this paper we have presented a solution to protect the AES against attacks of type SCA and FIA. The advantage of our solution compared to the state of the art is that it allows using the same structure to mask the sensitive information and at the same time to encode them. Moreover, the length of the mask does not directly depend on the masking order as shown in Table 2.

References

[1] Sabine Azzi, Bruno Barras, Maria Christofi, David Vigilant: Using linear codes as a fault countermeasure for nonlinear operations: application to AES and formal verification. J. Cryptographic Engineering 7(1): 75-85 (2017)

[2] Chari S., Jutla C.S., Rao J.R., Rohatgi P. (1999) Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener M. (eds) Advances in Cryptology — CRYPTO' 99. CRYPTO 1999. Lecture Notes in Computer Science, vol 1666.

[3] Goubin L., Patarin J. (1999) DES and Differential Power Analysis The "Duplication" Method. In: Koç Ç.K., Paar C. (eds) Cryptographic Hardware and Embedded Systems. CHES 1999. Lecture Notes in Computer Science, vol 1717.

[4] M. Rivain, E. Prouff, and J. Doget. (2009) Higher-order Masking and Shuffling for Software Implementations of Block Ciphers. Published in: Proceeding CHES '09 Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems Pages 171 - 188.

[5] Bringer J., Carlet C., Chabanne H., Guilley S., Maghrebi H. (2014) Orthogonal Direct Sum Masking. In: Naccache D., Sauveron D. (eds) Information Security Theory and Practice. Securing the Internet of Things. WISTP 2014. Lecture Notes in Computer Science, vol 8501. Springer, Berlin, Heidelberg

[6] Federal Information Processing Standards Publication 197 (2001). ADVANCED ENCRYPTION STANDARD (AES). November 26, 2001. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[7] Daemen, J., & Rijmen, V. (1999). AES proposal: Rijndael. ISO 690

[8] Balasch J., Faust S., Gierlichs B. (2015) Inner Product Masking Revisited. In: Oswald E., Fischlin M. (eds) Advances in Cryptology – EUROCRYPT 2015. EUROCRYPT 2015. Lecture Notes in Computer Science, vol 9056.

[9] Goubin L., Martinelli A. (2011) Protecting AES with Shamir's Secret Sharing Scheme. In: Preneel B., Takagi T. (eds) Cryptographic Hardware and Embedded Systems – CHES 2011. CHES 2011. Lecture Notes in Computer Science, vol 6917.

[10] Adi Shamir. 1979. How to share a secret. Commun. ACM 22, 11 (November 1979), 612-613. DOI=http://dx.doi.org/10.1145/359168.359176

[11] Prouff E., Roche T. (2011) Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In: Preneel B., Takagi T. (eds) Cryptographic Hardware and Embedded Systems – CHES 2011. CHES 2011. Lecture Notes in Computer Science, vol 6917. Springer, Berlin, Heidelberg.

[12] Ishai Y., Sahai A., Wagner D. (2003) Private Circuits: Securing Hardware against Probing Attacks. In: Boneh D. (eds) Advances in Cryptology - CRYPTO 2003. CRYPTO 2003. Lecture Notes in Computer Science, vol 2729. Springer, Berlin, Heidelberg

[13] Rivain M., Prouff E. (2010) Provably Secure Higher-Order Masking of AES. In: Mangard S., Standaert FX. (eds) Cryptographic Hardware and Embedded Systems, CHES 2010. CHES 2010. Lecture Notes in Computer Science, vol 6225. Springer, Berlin, Heidelberg

[14] Anderson R., Kuhn M. (1998) Low cost attacks on tamper resistant devices. In: Christianson B., Crispo B., Lomas M., Roe M. (eds) Security Protocols. Security Protocols 1997. Lecture Notes in Computer Science, vol 1361. Springer, Berlin, Heidelberg

[15] Boneh, D., DeMillo, R. & Lipton, R. J. Cryptology (2001) 14: 101. doi:10.1007/s001450010016

[16] A. Barenghi, L. Breveglieri, I. Koren and D. Naccache, "Fault Injection Attacks on Cryptographic Devices: Theory, Practice, and Countermeasures," in Proceedings of the IEEE, vol. 100, no. 11, pp. 3056-3076, Nov. 2012. doi: 10.1109/JPROC.2012.2188769

[17] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC '88). ACM, New York, NY, USA, 1-10. DOI=http://dx.doi.org/10.1145/62212.62213

[18] Massey, J. L. (1992). Linear codes with complementary duals. Discrete Mathematics, 106, 337-342.

[19] Bruneau N., Guilley S., Heuser A., Rioul O., Standaert FX., Teglia Y. (2016) Taylor Expansion of Maximum Likelihood Attacks for Masked and Shuffled Implementations. In: Cheon J., Takagi T. (eds) Advances in Cryptology – ASIACRYPT 2016. ASIACRYPT 2016. Lecture Notes in Computer Science, vol 10031. Springer, Berlin, Heidelberg