Detecting Faults in Inner Product Masking Scheme
IPM-FD: IPM with Fault Detection

Anonymous Submission

Abstract

Side-channel analysis and fault injection attacks are two typical threats to cryptographic implementations, especially in modern embedded devices. Thus there is an insistent demand for dual side-channel and fault injection protections. As it is known, masking scheme is a kind of provable countermeasure against side-channel attacks. Recently, inner product masking (IPM) was proposed as a promising higher-order masking scheme against side-channel analysis, but not for fault injection attacks. In this paper, we devise a new masking scheme named IPM-FD. It is built on IPM, which enables fault detection. This novel masking scheme has three properties: the security orders in the word-level probing model, bit-level probing model, and the number of detected faults. IPM-FD is proven secure both in the word-level and in the bit-level probing models, and allows for end-to-end fault detection against fault injection attacks.

Furthermore, we illustrate its security order by interpreting IPM-FD as a coding problem then linking it to one defining parameters of linear code, and show its implementation cost by applying IPM-FD to AES-128.

1 Introduction

With the advent of Internet of Things (IoT), more and more cryptographic libraries are implemented in software. Now, IoT objects are, most of the time, not made of secure hardware. Therefore, it is important for the software to protect itself in a sound manner. In this article, we assume that the implementation is free from configuration and coding bugs. Still, in this case, attackers can leverage two techniques to extract information: *side-channel* and *fault injection* analyses. Indeed, it is known that a single faulty encryption in AES can fully disclose 128 bits of the key [1]. It can be noted that some combined side-channel and fault analyses exist against protected implementations [8, 7].

On one hand, protections against Side-channel analysis aims at reducing the signal-to-noise ratio (see definition in [15, § 4.3.2]) an attacker can get. One option is to balance the leakage, technique which is used to linearize the control flow. For instance, cache-timing attacks can be alleviated by removing conditional opcodes whose condition is sensitive and sensitive pointer dereferencing. Besides, we assume Meltdown and ZombieLoad attack categories are irrelevant as the code we are interested in is baremetal. Still, there is the possibility of sensitive value leakage, which is properly addressed by randomization (*masking* [15, Chap. 9]). Indeed, sensitive values leak through a non-injective and noisy channel, thence single trace attacks are unpractical.

On the other hand, protections against fault injection attacks boil down to detection of errors, using either spacial, temporal, or information redundancy. Other techniques rely on invariant checking, such as idempotence of encryption composed by decryption.

In this paper, we present a joint countermeasure to both attacks, which is more efficient than two countermeasures piled one on top of each other.

State-of-the-art. In the scientific literature, early countermeasures against both side-channel and fault injection attacks have been designed in hardware. Several gate-level logic styles have been introduced, in particular dual-rail with precharge logic, aiming at balancing the leakage. Namely, redundant encodings, where each bit a is represented as a pair of bits (a_f, a_t), such that $a_f = \neg a_t = a$ during computation evaluation phase. Owing to this redundancy, the total number of bits set to 1 is unchanged (if in addition, the *evaluation* phase is interleaved with a *precharge* phase, the Hamming distance between two states is also constant, irrespective of the sensitive data manipulated). Besides, the redundant encoding $a_f = \neg a_t = a$ allows for computation checks, as in evaluation phase, $a_f = a_t$ (two configurations, namely $(0, 0)$ and $(1, 1)$) are forbidden. Starting from Wave Dynamic Differential Logic (WDDL [15, Chap. 7]), other improvements have been successively introduced (MDLP, iMDPL [12], ParTI [20], etc.) Also, some exotic styles have been proposed (asynchronous logic [17], adiabatic logic [16], etc.). All this corpus requires a hardware support.

In this paper, we target software-level countermeasures. We build upon the higher-order side-channel countermeasure known as IPM [2] to enrich it to detect faults injected during the computation.

Contributions. We devise an end-to-end fault-detection scheme which operates from within a provable high-order multivariate masking scheme. Practically, we enhance IPM scheme to enable end-to-end side-channel and fault injection detection, while keeping security proofs in the probing security model. Furthermore, we quantify the impact of both side-channel and fault detection on a complete AES cipher.

Outline. The rest of this paper is organized as follows. Sec. 2 introduce two typical schemes as the state-of-the-art of countermeasures. Our novel protection is presented in Sec. 3, with security analysis and optimal code selection in Sec. 4. The practical performance evaluation is presented in Sec. 5. Finally, Sec. 6 concludes the paper and opens some perspectives.

2 State-of-the-art on side-channel & fault protection

Side-channel protections come in two flavors:

- Inner Product Masking (IPM) [2] is a word-oriented (e.g., byte-oriented) masking scheme, equipped with universal operations (namely, addition and multiplication). It is optimized to resist attacks at both word-level and bit-level probing model, which is suitable for computing cryptographic algorithms that are subject to high-order side-channel analysis.

- Direct Sum Masking (DSM) [5] is a masking scheme which allows for concurrent side-channel and fault injection protection. It expresses the masking as the two encodings of the secret in a code C, and masks in a code D, respectively. This allows to recover the information by decoding from C and to check the masks by decoding from D.

These two protections are presented, one after the other, in this section.

2.1 Inner Product Masking

2.1.1 Notations

Computations are carried out in characteristic two finite fields: $\mathbb{F}_2$ for bits and $\mathbb{K}$ for larger fields. In practice $\mathbb{K}$ can be $\mathbb{F}_{2^l}$ for some l, e.g., $l = 8$ for AES, and $l = 4$ for PRESENT. The elements from $\mathbb{K}$ are termed *words*, and they are also referred to as *bytes* when $l = 8$ and to *nibbles* when $l = 4$. We denote $+$ the addition in characteristic two field $\mathbb{K}$, which is bitwise XOR. Recall that the subtraction is the same operation as the addition in $\mathbb{K}$. Elements of $\mathbb{F}_2$ are denoted as $\{0, 1\}$, and elements of $\mathbb{F}_{2^l}$ (as *words*) are represented as polynomials. In this paper, we use $\mathbb{F}_{2^4} \cong \mathbb{F}_2[\alpha]/\langle\alpha^4 + \alpha + 1\rangle$, and $\mathbb{F}_{2^8} \cong \mathbb{F}_2[\alpha]/\langle\alpha^8 + \alpha^4 + \alpha^3 + \alpha + 1\rangle$ (that of AES).

We recall that linear codes are spacevectors, characterized by their base field $\mathbb{K}$, their length n and their dimension k. In addition, linear codes have parameters traditionally denoted as $[n, k, d]$, where d is the minimum distance. The dual of a linear code D is the linear code $D^\perp$ whose codewords are orthogonal to all codewords of D. The dual distance $d^\perp$ of a code D happens to be equal to the minimum distance of $D^\perp$ [14].

Let n be the number of shares in IPM, and the coefficient vector in IPM is $\vec{L} = (L_1, L_2, \ldots, L_n)$ where $L_1 = 1$ for performance reason [2, § 1.2].

Definition 1 (IPM data representation)**.** *A word of secret information $X \in \mathbb{K}$ is represented in IPM as a tuple of n field elements:*

$$\vec{Z} = (X + \sum_{i=2}^{n} L_i M_i, M_2, \ldots, M_n) = X\mathbf{G} + \vec{M}\mathbf{H} \tag{1}$$

where $\vec{M} = (M_2, M_3, \ldots, M_n)$ is the mask materials, $\mathbf{G}$ *and* $\mathbf{H}$ *are generating matrices of linear codes C and D, respectively, as showed below.*

$$\mathbf{G} = \begin{pmatrix} 1 & | & 0 & 0 & \ldots & 0 \end{pmatrix} \quad \in \mathbb{K}^{1\times n} \tag{2}$$

$$\mathbf{H} = \begin{pmatrix} L_2 & | & 1 & 0 & \ldots & 0 \\ L_3 & | & 0 & 1 & \ldots & 0 \\ \vdots & | & 0 & 0 & \ddots & 0 \\ L_n & | & 0 & 0 & \ldots & 1 \end{pmatrix} \quad \in \mathbb{K}^{(n-1)\times n}. \tag{3}$$

The secret information X can be demasked by inner product between two vectors as: $X = \langle \vec{L}, \vec{Z} \rangle = \sum_{i=1}^{n} L_i Z_i$. Finally, we introduce some handy subset notations. Let $\vec{Z} = (Z_1, \ldots, Z_n) = (Z_i)_{i\in\{1,\ldots,n\}}$ a vector. We have:

$$\vec{Z}_I = (Z_i)_{i\in I} \quad \text{for} \quad I \subseteq \{1, \ldots, n\}.$$

For instance, $Z_{\{i\}\cup\{k+1,\ldots,n\}}$, for $1 \le i \le k \le n$, represents the $(n - k + 1)$ vector of coordinates $(Z_i, Z_{k+1}, \ldots, Z_n)$.

2.1.2 Security order regarding side-channel analysis

The security of IPM is stated in the *probing model* [11]: the security order is the maximum number of shares which are independent to masked information. We clarify word-level and bit-level security orders as follows:

- **Word-level (l-bit) security order $\boldsymbol{d_w}$:** since many devices perform computation on word-level data, especially on embedded devices byte-level operations are very common. In this paper, We also present instances for 4-bit (nibble) variables for adopting IPM to protect implementation of lightweight cipher like PRESENT, Simon and Speck, etc.
- **Bit-level security order $\boldsymbol{d_b}$:** in practice, each bit of sensitive variable can be investigated independently or/and several bits can be evaluated jointly. We consider here the number of bits that can be probed by attackers, which is consistent with the bit-level probing model proposed by Poussier *et al.* [18].

The main advantage of IPM is the higher bit-level security order than Boolean masking, which is called "Security Order Amplification" in [22]. It has been proven in [18] that side-channel resistance is directly connected to the dual distance $d_D^\perp$ of the code D generated by $\mathbf{H}$. Precisely, the security order t of IPM is equal to $t = d_D^\perp - 1$ [18].

The dual distance of linear code D is equal to the minimum distance of the dual code $D^\perp$. It is easy to see that the later has dimension 1 and is generated by $1 \times n$ matrix:

$$\mathbf{H}^\perp = \begin{pmatrix} 1 & L_2 & \dots & L_n \end{pmatrix}. \tag{4}$$

In order to investigate the bit-level security, the definition of expansion is introduced as follows.

Definition 2 (Code Expansion)**.** *By using sub-field representation, the elements in $\mathbb{K} = \mathbb{F}_{2^l}$ are decomposed into $\mathbb{F}_2$, we have:*

$$\mathit{SubfieldRepresentation}: \ (1, L_2, \dots, L_n)_{2^l} \longrightarrow (I_l, \mathbb{L}_2, \dots, \mathbb{L}_n)_2, \tag{5}$$

where I_l is the $l \times l$ identity matrix in $\mathbb{F}_2$ and $\mathbb{L}_i$ ($2 \le i \le n$) are $l \times l$ identity matrices.

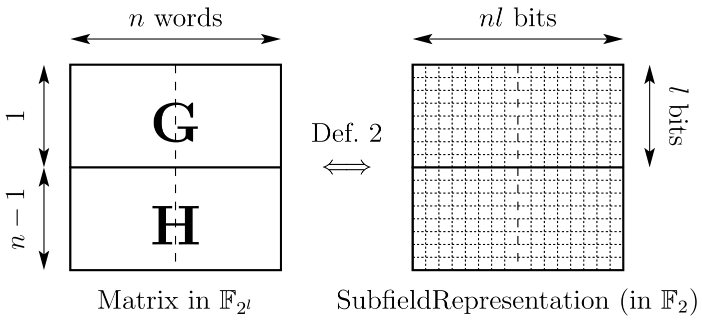
To derive the matrices, we can use that $\mathbb{F}_{2^l}$ is a field extension of $\mathbb{F}_2$, and given an irreducible polynomial P over $\mathbb{F}_2$ and denoting each element $a \in \mathbb{F}_{2^l}$ as $\sum_{i=0}^{l-1} a_i \alpha^i$ [mod $P(X)$], replace a by $(a_0, \dots, a_{l-1})$. Under the computer algebra system `Magma` [21], P is `DefiningPolynomial`$(\mathbb{F}_{2^l})$ and D' is `SubfieldRepresentationCode`(D). If D has parameters $[n, k, d]_{2^l}$, then D' has parameters $[nl, kl, d']_2$, where $d' \ge d$.

At word level, we notice that the dual distance of D is equal to n as long as $\forall i,\ L_i \neq 0$. As a result, word-level security orders of IPM is $d_w = n - 1$ [2]. In addition, security order d_b at the bit-level of IPM is equal to the dual distance of the code generated by the expansion of D from $\mathbb{F}_{2^l}$ to $\mathbb{F}_2$. A typical example of IPM codes matrices $\mathbf{G} = (1, 0)$ and $\mathbf{H} = (L_2, 1)$ in $\mathbb{F}_{2^8}$ is given in Fig. 1. The security order at word (byte) level is $d_w = n - 1 = 1$ and at bit level is $d_b = 3$ because the dual code of $D = \text{span}(H)$ is generated by $(1, L_2)$, which, after projection in $\mathbb{F}_2$, has parameters $[16, 8, 4]_2$.

Moreover, addition and multiplication are proven to be $t = (n-1)$-order secure at word-level in [3] using t-SNI property [4], thus the word-level security order is maintained by composition. Still, when a variable is reused, caution must be taken where a refresh algorithm is always adopted to avoid dependence. The refresh operation allows to decorrelate two copies of a variable that need to be used at two places (to avoid side-channel flaws as put forward in [9]). However, IPM cannot detect faults since no redundancy is inserted to the coding.

2.2 Direct Sum Masking

Direct sum masking has been originally introduced as Orthogonal Direct Sum Masking (ODSM [5]). The secret $\vec{X}$ is represented as a bitvector in $\mathbb{F}_2^l$. It is encoded using generating matrix $\mathbf{G}$ (of size $l \times nl$ in $\mathbb{F}_2$) as a word in $\mathbb{F}_2^{nl}$. Some random mask $\vec{M}$, drawn uniformly


n words
nl bits
1
$n-1$
G
H
Def. 2
$\Longleftrightarrow$
l bits
Matrix in $\mathbb{F}_{2^l}$
SubfieldRepresentation (in $\mathbb{F}_2$)

$$\mathbf{G} = \left(\begin{array}{c:c} 1 & 0 \end{array}\right) \quad \mathbf{G} = \left(\begin{array}{cccccccc:cccccccc}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0
\end{array}\right)$$

$$\mathbf{H} = \left(\begin{array}{c:c} L_2 = \\ \alpha^8 & 1 \end{array}\right) \quad \mathbf{H} = \left(\begin{array}{cccccccc:cccccccc}
1&0&0&0&1&1&0&1&1&0&0&0&0&0&0&0\\
1&1&0&0&1&0&1&1&0&1&0&0&0&0&0&0\\
0&1&1&0&0&1&0&1&0&0&1&0&0&0&0&0\\
1&0&1&1&1&1&1&1&0&0&0&1&0&0&0&0\\
1&0&0&1&0&0&1&0&0&0&0&0&1&0&0&0\\
0&1&1&0&1&0&0&1&0&0&0&0&0&1&0&0\\
0&1&1&1&0&1&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&1&0&1&0&0&0&0&0&0&0&0&1
\end{array}\right)$$

Figure 1: Dimensions (typical) of IPM encodings, for $n = 2$, on $l = 8$ bits

in $\mathbb{F}_2^{(n-1)l}$ is encoded with matrix $\mathbf{H}$ (of size $(n-1)l \times nl$). After masking the secret with the mask materials, one gets the protected information:

$$\vec{Z} = \vec{X}\mathbf{G} + \vec{M}\mathbf{H}. \tag{6}$$

On the contrary to IPM, the matrices $\mathbf{G}$ and $\mathbf{H}$ do not have specific form (recall IPM matrices are formatted as Eqn. 2 and 3). But there is no inverse operation of "SubfieldRepresentation" (recall Def. 2). Therefore, IPM is a special case of DSM, but some DSM encodings (Eqn. 6) cannot be represented as IPM.

ODSM uses orthogonal codes such that recovering $\vec{M}$ is straightforward knowing $\vec{Z}$: it consists in an orthogonal projection from spacevector $\mathbb{F}_2^n$ onto D. Actually, the complete commutative diagram involved in DSM is depicted in Fig. 2. The operations are explicited below:

- Information vector $\vec{X}$ is encoded as $\vec{X}\mathbf{G}$ (using linear application E_C), while decoding of $\vec{X}\mathbf{G}$ into $\vec{X}$ is ensured by the decoding application D_C;
- Similarly, masking random variables $\vec{M}$ are encoded as $\vec{M}\mathbf{H}$ (using linear application E_D). Decoding of $\vec{M}\mathbf{H}$ into $\vec{M}$ is ensured by the decoding application D_D;
- Creating an encoded word $\vec{Z}$ consists in adding one codeword $\vec{X}\mathbf{G}$ from C to one codeword $\vec{M}\mathbf{H}$ from D. In reverse, projections of $\vec{Z} \in \mathbb{K}^n$ to C (resp. D) is respectively obtained by linear projection operation Π_C (resp. Π_D).

When C and D are orthogonal, then $\mathbf{G}\mathbf{H}^\mathsf{T} = 0$, the all-zero $l \times (n-l)$ matrix. Therefore $\Pi_C(\vec{Z}) = \vec{Z}\mathbf{G}^\mathsf{T}(\mathbf{G}\mathbf{G}^\mathsf{T})^{-1}\mathbf{G}$ and $\Pi_D(\vec{Z}) = \vec{Z}\mathbf{H}^\mathsf{T}(\mathbf{H}\mathbf{H}^\mathsf{T})^{-1}\mathbf{H}$ [5].

This allows for the verification that an attacker which injects a fault has not corrupted the (useless in terms of exploitation) masks $\vec{M}$. In practice, the attack (addition of a nonzero bitvector $\epsilon \in \mathbb{F}_2^{nl}\backslash\{0\}$) is undetected if and only if $\epsilon \in C$. Indeed, otherwise ϵ has a nonzero component in spacevector D, and the fault injection is detected. The fault detection capability can be quantified in two models:

$$\begin{tikzcd}
\vec{X} \arrow[r, shift left, "E_C"] & \vec{X}\mathbf{G} \arrow[l, shift left, "D_C"] \arrow[dr] & & \\
& & + \arrow[r] & \vec{Z} = \vec{X}\mathbf{G} + \vec{M}\mathbf{H} \arrow[ull, "\Pi_C"'] \arrow[dll, "\Pi_D"] \\
\vec{M} \arrow[r, shift left, "E_D"] & \vec{M}\mathbf{H} \arrow[l, shift left, "D_D"] \arrow[ur] & &
\end{tikzcd}$$

Figure 2: Commutative diagram of DSM masking scheme

1. Assumption 1: the difficulty of the attack is all the larger as the number of bits flipped is larger. Thus, undetected faults $\epsilon \in C\backslash\{0\}$ must have Hamming weights $\geq d_C$, where d_C is the minimum distance of code C.

2. Assumption 2: the attacker can corrupt Z with the same difficulty whatever ϵ, but cannot control the value of ϵ. Said differently, ϵ is a random variable uniformly distributed in $\mathbb{F}_2^{nl}\backslash\{0\}$. The probability that the fault is not detected equals to $\frac{2^{l(n-1)}}{2^{nl}-1} \approx 2^{-l}$.

Thus, the probability of undetected faults gets lower as l increases. However, this approach has three drawbacks:

- First of all, the masks used in ODSM keep unchanged during each call of cipher, which allows fault detection. But the "static" masks may pose a vulnerability since masks should be refreshed to avoid unintended dependencies between sensitive variables.

- Secondly, it allows only to check errors on states $\vec{Z}$, but not during non-linear computations (which are tabulated, i.e., operations on $\vec{Z}$ consist in lookup table accesses). From a hardware point of view, this means that ODSM allows to detect faults in sequential logic (e.g., register banks, RAM, etc.), but not in combinational logic (e.g., logic gates or ROM).

- Thirdly, during verification, that is the projection of $\vec{Z} + \epsilon$ in spacevector D, the state $\vec{Z}$ is manipulated, hence additional leakage is produced, which must be taken into account in the security evaluation of DSM representation (Eqn. 6).

The first two points are structural weaknesses, and will be fixed in Alg. 3, starting from Section 3. For the third one, some codes suitable for DSM are constructed by Carlet *et al.* in [6] by duplicating the masks $\vec{M}$, while this solution do not allow an end-to-end scheme.

3 Novel end-to-end fault detection scheme

3.1 Rationale

The core idea in our new scheme is to duplicate (or more times) the secret X, rather than duplicating masks $\vec{M}$ as in [6], so that it can be checked at the end (when it is no longer sensitive–e.g., a ciphertext is a non-sensitive variable, so as the plaintext).

Our new scheme is a IPM-like masking scheme, called IPM-FD. Since IPM is a promising high-order masking scheme with higher bit-level security order, we extend it with fault detection capability so that it can resist side-channel analysis and fault injection attacks simultaneously. Specifically, we represent the information as a vector $(X_1, X_2, \ldots, X_k) \in \mathbb{K}^k$ where $\mathbb{K} = \mathbb{F}_{2^l}$.

We propose the new encoding as follows. Let us denote:

Definition 3 (IPM-FD data representation)**.**

$$\vec{Z} = \underbrace{(X_1, X_2, \ldots, X_k)}_{secrets\ \vec{X}}\ \mathbf{G} + \underbrace{(M_{k+1}, \ldots, M_n)}_{masks\ \vec{M}}\ \mathbf{H} = (Z_1, Z_2, \ldots, Z_n) \tag{7}$$

where

$$\mathbf{G} = (I_k||0) \in \mathbb{K}^{k\times n}$$
$$\mathbf{H} = (L||I_{n-k}) \in \mathbb{K}^{(n-k)\times n}$$

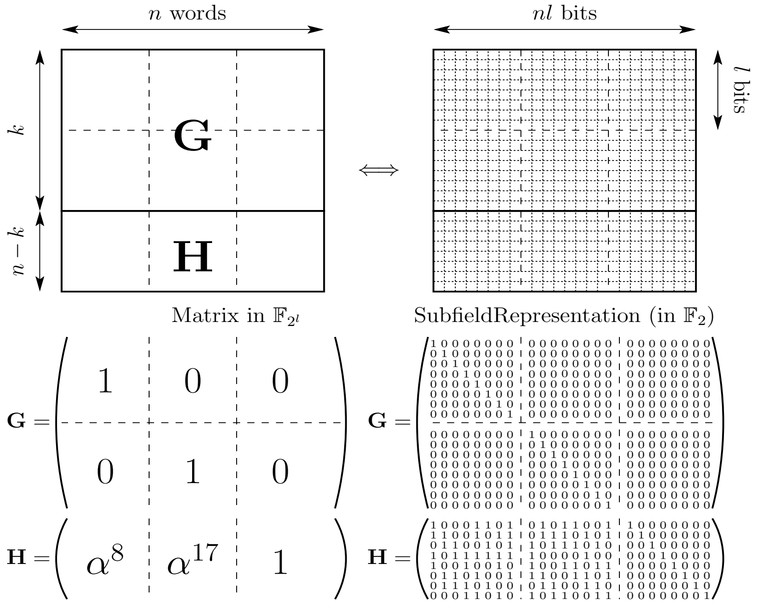
here I_k is the identity matrix $k \times k$ in $\mathbb{K}$, and L is a matrix of size $(n-k)\times k$, that is L has coefficients $(L_{i,j})_{k<i\leq n, 1\leq j\leq k}$.

This definition 3 is a generalization of Def. 1. In practice, we will call Equ. 7 with redundancy to detect faults in the information X, i.e., $(X_1, X_2, \ldots, X_k) = (X, X, \ldots, X)$ as:

$$\vec{Z} = \underbrace{(X, X, \ldots, X)}_{k \text{ times}}\mathbf{G} + (M_{k+1}, \ldots, M_n)\mathbf{H}. \tag{8}$$

For the sake of convenience, the IPM-FD encoding used in this paper is depicted in Fig. 3. It illustrates a protection using $n = 3$ shares of $l = 8$ bits, with those security features:

- $d_w = 1$ (1st-order secure at byte-level), because dual distance of $\mathbf{H}$ in $\mathbb{F}_{2^8}$ is 2;
- $d_b = 3$ (3rd-order secure at bit-level), since dual distance of $\mathbf{H}$ in $\mathbb{F}_2$ is 4—the subfield representation of code spawn by $\begin{pmatrix}1 & L_2 & L_3\end{pmatrix}$ has parameters $[24, 8, 4]_2$ (showed in Fig. 3).



n words
nl bits
k
$n-k$
l bits
G
H
⟺
Matrix in $\mathbb{F}_{2^l}$
SubfieldRepresentation (in $\mathbb{F}_2$)
$\mathbf{G} =$
$\mathbf{H} =$
1 0 0
0 1 0
α^8 α^{17} 1

Figure 3: Dimensions (typical) of IPM-FD encodings, for $n = 3$, $k = 2$ and $l = 8$ bits

Computation can be carried out on such $\vec{Z}$, and when it is over (e.g., the complete AES is finished), the implementation can check whether the k copies of the information are the same.

This allows to detect up to $(k-1)$ errors (there is an error if the k copies are not equal to each other). This end-to-end computation is the speciality of this approach. On the contrary, approach proposed on [6] needs to check for faults at each step of the computation.

Repeating X for k times may increase the signal captured by the attacker by a factor k. Nonetheless, such aspect is irrelevant to security order. Notice that, as future extension, one might consider an encoding of information X which is more efficient in terms of rate than the simple k-times repetition code $X \mapsto (X, \ldots, X)$. However, such representation in Equ. 8 allows for an end-to-end security protection against fault injection attacks, as illustrated in Alg. 3.

For fault detection, either the algorithm 3 is started from scratch, or other actions, such as event logging for subsequent analysis (aiming at taking proactive actions to plug this leak). It is obvious that detecting fault in each intermediate can be carried out at any place in Alg. 3, especially during steps 5. However, such precaution is superfluous, as an overall check is done at the end, that is at line 8. In addition, intermediate checks would disclose when the fault occurs (e.g., at which round), which delivers precious feedback to the attacker regarding the accuracy and the reproducibility of the setups.

3.2 Computing with representation of IPM-FD (as in Def. 3)

First of all, we present one instance of IPM-FD with $k=2$ to clarify its encoding. We denote $\vec{X} = (X_1, X_2) \in \mathbb{K}^2$, and $\vec{M} = (M_3, \ldots, M_n) \in \mathbb{K}^{n-2}$. Thus, we have Eqn. 7 such that,

$$\mathbf{G} = \begin{pmatrix} 1 & 0 & | & 0 & 0 & \ldots & 0 \\ 0 & 1 & | & 0 & 0 & \ldots & 0 \end{pmatrix}$$

$$\mathbf{H} = \begin{pmatrix} L_{3,1} & L_{3,2} & | & 1 & 0 & \ldots & 0 \\ L_{4,1} & L_{4,2} & | & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & | & 0 & 0 & \ddots & 0 \\ L_{n,1} & L_{n,2} & | & 0 & 0 & \ldots & 1 \end{pmatrix}$$

or, said differently, we have $\vec{Z} = (Z_1, \ldots, Z_n) \in \mathbb{K}^n$ which is equal to:

$$\begin{aligned} Z_1 &= X_1 + L_{3,1}M_3 + L_{4,1}M_4 + \ldots + L_{n,1}M_n \\ Z_2 &= X_2 + L_{3,2}M_3 + L_{4,2}M_4 + \ldots + L_{n,2}M_n \\ Z_i &= M_i \qquad \text{for } 3 \leq i \leq n \end{aligned}$$

Here, we can see that $(Z_1, Z_3, \ldots, Z_n) \in \mathbb{K}^{n-1}$ and $(Z_2, Z_3, \ldots, Z_n) \in \mathbb{K}^{n-1}$ are two IPM [2] sharings. Therefore, we have $k=2$ ways to demask:

$$\langle L_1, \vec{Z} \rangle = X_1 = X, \quad \text{and} \quad \langle L_2, \vec{Z} \rangle = X_2 = X,$$

where as a convention, $L_{1,1} = L_{2,2} = 1$, $L_{1,2} = L_{2,1} = 0$ and:

$$L_1 = (L_{i,1})_{1 \leq i \leq n} \in \mathbb{K}^n, \quad \text{and} \quad L_2 = (L_{i,2})_{1 \leq i \leq n} \in \mathbb{K}^n.$$

It is known that universal computation can be achieved by Lagrange interpolation, which requires addition and multiplication. Hereafter, we present two basic algorithms (also refresh algorithm in appendix. A), with the most general case (k words of information and scalable with different k) used to build a complete masked cryptographic implementation.

Algorithm 1: Secure addition in IPM-FD

```
input : Two sets of scalar tuples $\vec{X} = (X_1, \ldots, X_k)$ and $\vec{X'} = (X'_1, \ldots, X'_k)$ shared as:
    • $\vec{Z} = (Z_1, \ldots, Z_n) = (X_1 + \sum_{i=k+1}^{n} L_{i,1} M_i, \ldots, X_k + \sum_{i=k+1}^{n} L_{i,k} M_i, M_{k+1}, \ldots, M_n) \in \mathbb{K}^n$,
    • $\vec{Z'} = (Z'_1, \ldots, Z'_n) = (X'_1 + \sum_{i=k+1}^{n} L_{i,1} M'_i, \ldots, X'_k + \sum_{i=k+1}^{n} L_{i,k} M'_i, M'_{k+1}, \ldots, M'_n) \in \mathbb{K}^n$.
output: A sharing $\vec{R} = (R_1, \ldots, R_n) \in \mathbb{K}^n$ such that, for all $j$ $(1 \leq j \leq k)$,
        $\langle \vec{R}_{\{j\} \cup \{k+1,\ldots,n\}}, \vec{L}_{\{j\} \cup \{k+1,\ldots,n\},j} \rangle = X_j + X'_j$
1 $\vec{R} = (Z_1 + Z'_1, \ldots, Z_n + Z'_n)$
2 return $R$
```

3.2.1 Secure addition of IPM-FD

With Equ. 8, we denote encoding of X and X' by $\vec{Z}$ and $\vec{Z'}$ respectively, thus the addition is linear and can be calculated straightforwardly as in Alg. 1.

3.2.2 Secure multiplication of IPM-FD

Secure multiplication can be achieved by selecting only one amongst the k first coordinates, while keeping the $(n-k)$ masks, and multiplying $(n-k+1)$ shares by using the original IPM multiplication. Therefore, multiplication of IPM-FD is implemented in Alg. 2.

Algorithm 2: Secure multiplication of IPM-FD with k pieces of information

```
input : Two sets of scalar tuples $\vec{X} = (X_1, \ldots, X_k)$ and $\vec{X'} = (X'_1, \ldots, X'_k)$ shared as:
    • $\vec{Z} = (Z_1, \ldots, Z_n) = (X_1 + \sum_{i=k+1}^{n} L_{i,1} M_i, \ldots, X_k + \sum_{i=k+1}^{n} L_{i,k} M_i, M_{k+1}, \ldots, M_n) \in \mathbb{K}^n$,
    • $\vec{Z'} = (Z'_1, \ldots, Z'_n) = (X'_1 + \sum_{i=k+1}^{n} L_{i,1} M'_i, \ldots, X'_k + \sum_{i=k+1}^{n} L_{i,k} M'_i, M'_{k+1}, \ldots, M'_n) \in \mathbb{K}^n$.
output: A sharing $\vec{P} = (P_1, \ldots, P_n) \in \mathbb{K}^n$ such that, for all $j$ $(1 \leq j \leq k)$,
        $\langle \vec{P}_{\{j\} \cup \{k+1,\ldots,n\}}, \vec{L}_{\{j\} \cup \{k+1,\ldots,n\},j} \rangle = X_j \cdot X'_j$
1 for $j \in \{1, \ldots, k\}$ do
2     $\vec{P}[j] \leftarrow \mathsf{IPMult}(Z_{\{j\} \cup \{k+1,\ldots,n\}}, Z'_{\{j\} \cup \{k+1,\ldots,n\}})$        // IPMult is Alg. 5 of [2]
3     Let us write $\vec{P}[j]$ as $(P_j, N_{k+1,j}, \ldots, N_{n,j})$, where $P_j = X_j X'_j + \sum_{i=k+1}^{n} L_{i,j} N_{i,j} \in \mathbb{K}$
4 for $j \in \{2, \ldots, k\}$ do                    // Masks homogenization between $\vec{P}[1]$ and $\vec{P}[j]$
5     for $i \in \{k+1, \ldots, n\}$ do
6         $P_j \leftarrow P_j + L_{i,j}(N_{i,1} + N_{i,j})$
      // $(P_j, N_{k+1,1}, \ldots, N_{n,1})$ is a sharing of $X_j X'_j$ by the $(n-k)$ masks of $\vec{P}[1]$
7 return $\vec{P} = (P_1, \ldots, P_k, N_{k+1,1}, \ldots, N_{n,1}) \in \mathbb{K}^n$.
```

k times of multiplication are thus carried out on shares in $\mathbb{K}^{n-k+1}$, and the resulting $\vec{P}[j] \in \mathbb{K}^{n-k+1}$ for $j \in \{1, \ldots, k\}$ are applied from line 4 to line 6 as in Alg. 2 to homogenize masks in $(k-1)$ sharings with the same masks as $\vec{P}[1]$.

We call from line 4 to line 6 in Alg. 2 as homogenization algorithm used to merge the results

$\vec{P}[j]$ where $1 \leq j \leq k$. Thus we have the following lemma, which applies to non-redundant sharings such as that of Eqn. 1.

Lemma 1 (Homogenization of two sharings)**.** *Let $\vec{Z} = (Z_1, \dots, Z_n)$ and $\vec{Z}' = (Z'_1, \dots, Z'_n)$ be two sharings, that $\langle L, \vec{Z} \rangle = X$ and $\langle L', \vec{Z}' \rangle = X'$. There exists an equivalent sharing $\vec{Z}''$ and an algorithm to transform $\vec{Z}'$ into $\vec{Z}''$ such that $\vec{Z}$ and $\vec{Z}''$ share all coordinates but the first one.*

The proof of Lemma 1 is given in appendix B. The full masked implementation of a cryptographic algorithm can be built as in Alg. 3. Therefore, one can start with plaintext & key representation as Equ. 8 and carry addition / multiplication to implement cryptographic algorithms like AES, and end up with a ciphertext still with the form as Equ. 8. Hence verification can be done only at the very end.

Algorithm 3: End-to-end protection of a cryptographic algorithm (here AES-128) against fault injection attacks using IPM-FD scheme

```
input  : Plaintext $X \in \mathbb{F}_{2^8}^{16}$, key $K \in \mathbb{F}_{2^8}^{16}$, and number of detected faults $d_f = k - 1$,
         number of shares $n = d_w + 1$, bit-level security order $d_b = d_D^{\perp} - 1$
output: Ciphertext, or $\perp$ if a fault has been detected
The matrices $\mathbf{G}$ and $\mathbf{H}$ (corresponding to code $C$ and $D$, respectively) are determined
  with respect to the requirements on side-channel and fault protection $d_w$, $d_b$ and $d_f$ (see
  examples in Tab. 1 and Tab. 3)
$\vec{M} \leftarrow_{\mathcal{R}} \mathbb{F}_{2^8}^{16\times(n-k)}$
$\vec{Z} \leftarrow (X, \dots, X)\mathbf{G} + \vec{M}\mathbf{H}$                    // Recall Eqn. 8
...
Arithmetic operations for the (secure) computation, using Lagrange interpolation
  polynomial. This includes additions (Alg. 1) and multiplications (Alg. 2)
...
$(X_1, \dots, X_k) \leftarrow \Pi_C(\vec{Z})$                    // Recall $\Pi_C(\vec{Z})$ in Fig. 2
if $X_1 = \ldots = X_k$ then
    return $X_1$
else
    return $\perp$
```

4 Security analysis of IPM-FD and optimal codes selection

The security level of IPM-FD can be characterized by three metrics, namely word-level security order d_w, bit-level security order d_b and number of detected faults d_f $(=k-1)$. In this section, we show the security order of IPM-FD and how to choose optimal codes by interpreting IPM-FD as a coding problem.

4.1 Security order of IPM-FD on SCA resistance

The addition and refresh algorithm are secure since there is no degradation on masks, we focus on multiplication algorithm Alg. 2 and we have following theorem 1.

Theorem 1. *The multiplication of IPM-FD in Alg. 2 is $d_D^{\perp} - 1$ order secure.*

Proof. The k IPMult multiplications at line 2 are secure at $(n-k)$-th order [2]. After their application, the k shared variables $\vec{P}[j]$, $1 \leq j \leq k$, are masked by $N_{i,j}$ $(k{+}1 \leq i \leq n, 1 \leq j \leq k)$ that are $(n-k) \times k$ uniformly distributed and i.i.d. random variables.

At step 6, indexed by i $(k+1 \leq i \leq n)$, the contents of P_j is:

$$P_j = X_j X'_j + \left(\sum_{i'=k+1}^{i} L_{i',j} N_{i',1} \right) + \left(\sum_{i'=i+1}^{n} L_{i',j} N_{i',j} \right). \tag{9}$$

It is easy to see that any collusion of intermediate variations with mixed variables masked by $N_{i,j}$ and $N_{i,j'}$, for $j \neq j'$, requires more intermediate values to be probed than strategies which focus on a given $N_{i,j}$ (for a given j).

The intermediate variables which are *pure* in $P_{i,1}$ (since homogenization process consists in turning $N_{i,j}$ into $N_{i,1}$) are those at:

- line 2: $X_1 X'_1 + \sum_{i=k+1}^{n} L_{i,1} N_{i,1}$, and the $(n-k)$ masks $N_{i,1}$ $(k+1 \leq i \leq n)$;
- line 6: for $i = n$, $P_j = X_j X'_j + \sum_{i=k+1}^{n} L_{i,j} N_{i,1}$.

Those *pure* shares are those of results $\vec{P}$ (line 7). Together, they have the shape:

$$\vec{P} = (X_1, \ldots, X_k)\, \mathbf{G} + \vec{N}\, \mathbf{H},$$

where $\vec{N} = (N_{k+1,1}, \ldots, N_{n,1}) \in \mathbb{K}^{n-k}$ is a uniformly distributed tuple of i.i.d random variables. Hence $d_D^{\perp} - 1$ columns of $\mathbf{H}$ are independent [13, Theorem 10], which means if the attacker probes up to $d \leq (d_D^{\perp} - 1)$ variables, the secret X_j encoded as an element of $\mathbb{F}_{2^l}^{n-k+1}$ is perfectly masked. The security order of Alg. 2 is $(d_D^{\perp} - 1)$. □

In summary, the security order at word-level d_w and bit-level d_b of IPM-FD corresponding to $(d_D^{\perp} - 1)$ at word-level and bit-level (by Code Expansion defined in Def. 2), respectively. In particular, the maximum word-level security order d_w is $(n-k)$, since $d_D^{\perp} \leq (n-k+1)$ with equal if and only if $d_D^{\perp}$ is maximized.

4.2 Choosing optimal codes for IPM-FD

Two security orders d_w and d_b are connected to dual distance of D at word-level and bit-level, by encoding Eqn. 7 and Eqn. 8. Thus, we can search for minimal n satisfying the given requirements on the three parameters d_f, d_w and d_b. Since the best d_b is much difficult to obtain, we first search for codes given d_f and d_w, then find the best one with respect to optimal d_b. For the first step, the Alg. 4 is adopted for selecting codes with minimal n given d_f and d_w.

Algorithm 4: Selecting codes given d_f and d_w.

```
input : $l$ for $\mathbb{K} = \mathbb{F}_{2^l}$, $d_f$ for number of detected faults and $d_w$ for word-level
        side-channel security
output: the minimal $n$ satisfying the requirements
$n = d_w$    // $n$ is at least the minimum distance of the code generated by $\mathbf{H}^{\perp}$
while $MinimumDistance([BKLC(GF(2^l), n, d_f + 1)] < d_w)$ do
    $n{+}{+}$
return $n$
```

The second step is to choose the best code with maximal bit-level security d_b. We propose Alg. 5 to select optimal codes with maximized d_b.

Algorithm 5: Choosing optimal codes with maximal d_b.

```
input : $l$ for $\mathbb{K} = \mathbb{F}_{2^l}$, $d_f$ for number of detected faults, $d_w$ for word-level side-channel
        security and number of shares $n$
output: the maximal $d_b$ and optimal code $D$
$d_b = d_w$                    // Security order at bit-level is greater than word-level
$D_{opt} = null$
forall the code $D = [n, d_f + 1, d_w + 1]_{2^l}$ do
    $D_2$ = SubfieldRespresentation( $D$, $GF(2)$ )
    if $d_b <$ MinimumDistance($D_2$) then
        $d_b$ = MinimumDistance($D_2$)
        $D_{opt} = D$
return $d_b$, $D_{opt}$
```

We present some examples for codes in $\mathbb{F}_{2^8}$ in Tab. 1 (for $\mathbb{F}_{2^4}$ in Tab. 3, resp) calculated by `Magma` for small k and n. Interestingly, we compare the original IPM and IPM-FD with n and $n+1$ shares respectively, since in IPM-FD redundancy is needed for fault detection. For IPM with $n = 3$, we have optimal parameters $d_w = 2$ and $d_b = 5$, while for IPM-FD with $n = 4$, $k = 2$, the optimal d_w and d_b are $d_w = 2$ and $d_b = 4$. Hence there is a trade-off for fault detection, which sacrifices the bit-level side-channel resistance.

For instance, for $k = 2$, we can detect one error. We recall that the security order of IPM at bit level is given by the minimum distance of the code generated by $\mathbf{H}^\perp = (1, L_2, \ldots, L_n)$ (projected from $\mathbb{K} = \mathbb{F}_{2^l}$ to the binary ground field $\mathbb{F}_2^l$). Now, adding fault detection capability, the security order of IPM-FD becomes that of the minimum distance of the code generated by Equ. 10. However, the minimum distance of this code is less than that generated by either: $(1, L_{3,1}, L_{4,1}, \ldots, L_{n,1})$ or $(1, L_{3,2}, L_{4,2}, \ldots, L_{n,2})$.

$$\mathbf{H'}^\perp = \begin{pmatrix} 1 & 0 & L_{3,1} & L_{4,1} & \ldots & L_{n,1} \\ 0 & 1 & L_{3,2} & L_{4,2} & \ldots & L_{n,2} \end{pmatrix}. \tag{10}$$

5 Practical implementation and performances

We implement IPM-FD scheme on AES-128. All the computations are made with additions, multiplications and lookups in some pre-computed tables.The random number generator comes from the Sodium library [10]. Each sensitive variable ($Nb * (Nr + 1)$ subkeys from the *Key Schedule* routine and Nb bytes in state array), is masked into n shares using $n - k$ random bytes. In particular, about non-linear operations, the S-box of a masked value is computed using instead of the 256-sized table, its polynomial expression obtained via Lagrange interpolation:

$$x \in \mathbb{F}_{2^8} \mapsto \mathtt{63} + \mathtt{8f}x^{127} + \mathtt{b5}x^{191} + \mathtt{01}x^{223} + \mathtt{f4}x^{239} + \mathtt{25}x^{247} + \mathtt{f9}x^{251} + \mathtt{09}x^{253} + \mathtt{05}x^{254}.$$

After demasking a shared variable, we check that the data has not been faulted by comparing the k copies and pose an alarm if any fault is detected. Our implementation works for any $n > k$. Specially, for $n < 5$ and $k < 3$ we choose the Best Known Linear Code (BKLC) D obtained with `Magma` otherwise we randomly generate a matrix for masking.

[1] The maximal d_b for IPM codes with $n = 4$ shares in $\mathbb{F}_{2^8}$ is only 10 ($d_D^\perp$ equals 11), not 11 as in [18]

Table 1: Instances of codes with $X \in \mathbb{K} = \mathbb{F}_{2^8}$, d_b in IPM entries are consist with results in [18]

	Inputs		Outputs of Alg. 4 and Alg. 5		
	#faults d_f	d_w	n	d_b	Setting
IPM	0	0	1	0	$\mathbf{H}^{\perp} = \begin{pmatrix} 1 \end{pmatrix}$
	0	1	2	3	$\mathbf{H}^{\perp} = \begin{pmatrix} 1 & X^8 \end{pmatrix}$
	0	2	3	7	$\mathbf{H}^{\perp} = \begin{pmatrix} 1 & X^8 & X^{26} \end{pmatrix}$
	0	3	4	10[1]	$\mathbf{H}^{\perp} = \begin{pmatrix} 1 & X^8 & X^{26} & X^{17} \end{pmatrix}$
IPM-FD	1	0	2	0	$\mathbf{H}^{\perp} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
	1	1	3	3	$\mathbf{H}^{\perp} = \begin{pmatrix} 1 & 0 & X^8 \\ 0 & 1 & X^{17} \end{pmatrix}$
	1	2	4	6	$\mathbf{H}^{\perp} = \begin{pmatrix} 1 & 0 & X^8 & X^{20} \\ 0 & 1 & X^{27} & X^7 \end{pmatrix}$

5.1 Performance evaluation

We make a comparison for the same security order at word-level, between:

- No fault detection (classic IPM, $k = 1$) – this is our reference
- Single fault detection by temporal redundancy (repeat twice the IPM computation)
- Single fault detection embedded into IPM (so-called IPM-FD for $k = 2$)

Performance-wise, Tab. 2 shows that two fault detection strategies (temporal repetition and IPM-FD) have almost identical performances. But if we consider the most time-consuming operation, the field multiplication, the number of field multiplications in IPM on n shares (Alg. 5 of [2]) is $3n^2 - n$. While the number of multiplications in IPM-FD on n shares is:

Table 2: Performance comparison of IPM-FD without and with faults detection. *random* is the number of generated random bytes when masking and refreshing, *speed* is the runtime in milliseconds obtained on a virtual machine running with 2.8 GHz 6-core processor.

Security order	**IPM (baseline)**	**Two consecutive executions of IPM**	**IPM-FD** $k = 2$
$d_w = 1$	$n = 2$ ($d_b = 3$), speed= 1.52, random=1936	$n = 2$ ($d_b = 3$), speed= 3.04, random=3872	$n = 3$ ($d_b = 3$), speed= 2.93, random=3856
$d_w = 2$	$n = 3$ ($d_b = 7$), speed= 2.25, random= 5152	$n = 3$ ($d_b = 7$), speed= 4.50, random= 10304	$n = 4$ ($d_b = 6$ — we loose 1 order at bit-level), speed= 4.31, random= 10272

- $k(3(n-k+1)^2 - (n-k+1))$ regarding the k IPM multiplications on $n-k+1$ shares,
- $(k-1)(n-k)$ regarding the $(k-1)$ homogenizations.

Hence a total complexity of $k(3(n-k+1)^2 - (n-k+1)) + (k-1)(n-k)$, that is:

- $3n^2 - n$ for IPM-FD with $k = 1$,

- $6n^2 - 13n + 6$ for IPM-FD with $k = 2$.

Now, we have that $2\times(3n^2-n) > 6n^2-13n+6$, therefore it is more interesting, complexity-wise, to use IPM-FD for $k = 2$ than to repeat twice a computation.

Notice that temporal redundancy is prone to fault injection attacks, whereby an attacker would reproduce exactly the same fault on the repeated executions. Therefore, method IPM-FD is intrinsically stronger against fault attacks, at the same cost in terms of execution speed.

6 Conclusion and perspectives

IPM shows an advantageous property - higher security order at bit-level d_b than at work-level - as a promising alternative to Boolean masking. In this paper, we propose a novel end-to-end fault detection scheme called IPM-FD, which is a IPM-like scheme to detect faults by redundancy on secrets rather than on masks. The IPM-FD is also a unified scheme to resist side-channel analysis and fault injection attack simultaneously. We also present an instance by applying IPM-FD to AES and provide a comparison on performance with different settings.

As a perspective, we notice that the performances of both IPM and IPM-FD can be improved by choosing small (or sparse) values for $L_{i,j} \in \mathbb{K}$ scalars. This strategy is similar to that already employed by Rijndael inventors, for instance when designing the MixColumns operation. This raises the question of finding codes with sparse matrices of high dual distance.

References

[1] Subidh Ali, Debdeep Mukhopadhyay, and Michael Tunstall. Differential fault analysis of AES: towards reaching its limits. *J. Cryptographic Engineering*, 3(2):73–97, 2013.

[2] Josep Balasch, Sebastian Faust, and Benedikt Gierlichs. Inner product masking revisited. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 486–510. Springer, 2015.

[3] Josep Balasch, Sebastian Faust, Benedikt Gierlichs, Clara Paglialonga, and François-Xavier Standaert. Consolidating Inner Product Masking. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 724–754. Springer, 2017.

[4] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, and Benjamin Grégoire. Compositional Verification of Higher-Order Masking: Application to a Verifying Masking Compiler. *IACR Cryptology ePrint Archive*, 2015:506, 2015.

[5] Julien Bringer, Claude Carlet, Hervé Chabanne, Sylvain Guilley, and Houssem Maghrebi. Orthogonal Direct Sum Masking - A Smartcard Friendly Computation Paradigm in a Code, with Builtin Protection against Side-Channel and Fault Attacks. In David Naccache and Damien Sauveron, editors, *Information Security Theory and Practice. Securing the Internet of Things - 8th IFIP WG 11.2 International Workshop, WISTP 2014, Heraklion, Crete, Greece, June 30 - July 2, 2014. Proceedings*, volume 8501 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2014.

[6] Claude Carlet, Cem Güneri, Sihem Mesnager, and Ferruh Özbudak. Construction of some codes suitable for both side channel and fault injection attacks. In Lilya Budaghyan and Francisco Rodríguez-Henríquez, editors, *Arithmetic of Finite Fields - 7th International Workshop, WAIFI 2018, Bergen, Norway, June 14-16, 2018, Revised Selected Papers*, volume 11321 of *Lecture Notes in Computer Science*, pages 95–107. Springer, 2018.

[7] Abhishek Chakraborty, Bodhisatwa Mazumdar, and Debdeep Mukhopadhyay. A combined power and fault analysis attack on protected grain family of stream ciphers. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 36(12):1968–1977, 2017.

[8] Christophe Clavier, Benoît Feix, Georges Gagnerot, and Mylène Roussellet. Passive and Active Combined Attacks on AES. In *FDTC*, pages 10–18. IEEE Computer Society, 21 August 2010. Santa Barbara, CA, USA. DOI: 10.1109/FDTC.2010.17.

[9] Jean-Sébastien Coron, Emmanuel Prouff, and Matthieu Rivain. Side Channel Cryptanalysis of a Higher Order Masking Scheme. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES*, volume 4727 of *LNCS*, pages 28–44. Springer, 2007.

[10] Frank Denis. The sodium cryptography library, Jul 2019.

[11] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, August 17–21 2003. Santa Barbara, California, USA.

[12] Mario Kirschbaum and Thomas Popp. Evaluation of a DPA-Resistant Prototype Chip. In *ACSAC*, pages 43–50. IEEE Computer Society, 7-11 December 2009. Honolulu, Hawaii.

[13] F. Jessie MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*. Elsevier, Amsterdam, North Holland, 1977. ISBN: 978-0-444-85193-2.

[14] Florence Jessie MacWilliams and N. J. A. Neil James Alexander Sloane. *The theory of error correcting codes*. North-Holland mathematical library. North-Holland Pub. Co. New York, Amsterdam, New York, 1977. Includes index.

[15] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006. ISBN 0-387-30857-1, http://www.dpabook.org/.

[16] Cancio Monteiro, Yasuhiro Takahashi, and Toshikazu Sekine. Low power secure aes s-box using adiabatic logic circuit. In *2013 IEEE Faible Tension Faible Consommation*, pages 1–4, June 2013.

[17] Simon Moore, Ross Anderson, Robert Mullins, George Taylor, and Jacques J.A. Fournier. Balanced Self-Checking Asynchronous Logic for Smart Card Applications. *Journal of Microprocessors and Microsystems*, 27(9):421–430, October 2003.

[18] Romain Poussier, Qian Guo, François-Xavier Standaert, Claude Carlet, and Sylvain Guilley. Connecting and Improving Direct Sum Masking and Inner Product Masking. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, volume 10728 of *Lecture Notes in Computer Science*, pages 123–141. Springer, 2017.

[19] Matthieu Rivain and Emmanuel Prouff. Provably Secure Higher-Order Masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.

[20] Tobias Schneider, Amir Moradi, and Tim Güneysu. Parti - towards combined hardware countermeasures against side-channel and fault-injection attacks. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2016.

[21] University of Sydney. Magma Computational Algebra System. http://magma.maths.usyd.edu.au/magma/, Accessed on 2014-08-22.

[22] Weijia Wang, François-Xavier Standaert, Yu Yu, Sihang Pu, Junrong Liu, Zheng Guo, and Dawu Gu. Inner Product Masking for Bitslice Ciphers and Security Order Amplification for Linear Leakages. In Kerstin Lemke-Rust and Michael Tunstall, editors, *Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, November 7-9, 2016, Revised Selected Papers*, volume 10146 of *Lecture Notes in Computer Science*, pages 174–191. Springer, 2016.

A Secure refresh algorithm for IPM-FD

As pointed in [19], we need to apply a refresh algorithm after each squaring operation to keep independence between masks (Alg. 2 with $\vec{Z} = \vec{Z}'$). The algorithm for the refresh of IPM-FD is given in Alg. 6. Notice that this algorithm can be computed in-place, meaning that the output overwrites the input.

Algorithm 6: IPM-FD refresh algorithm

```
input : Let $k < n$. One IPM-FD sharing $\vec{Z} = (X_1, \dots, X_k)\,\mathbf{G} + (M_{k+1}, \dots, M_n)\,\mathbf{H}$, as
        defined in Equ. 7
output: An equivalent IPM-FD sharing $\vec{Z}' = (X_1, \dots, X_k)\,\mathbf{G} + (M'_{k+1}, \dots, M'_n)\,\mathbf{H}$,
        where $(M_{k+1}, \dots, M_n)$ is independent from $(M'_{k+1}, \dots, M'_n)$.

$\vec{Z}' \leftarrow \vec{Z}$                          // When computed in-place, $\vec{Z}'$ is not needed.
for $i \in \{k+1, \dots, n\}$ do
    $\epsilon \leftarrow_{\mathcal{R}} \mathbb{K}$                              // Fresh random variable
    $Z'_i \leftarrow Z'_i + \epsilon$
    for $j \in \{1, \dots, k\}$ do
        $Z'_j \leftarrow Z'_j + L_{i,j}\epsilon$
return $\vec{Z}' \in \mathbb{K}^n$.
```

B Proof of Lemma 1

Recall that Lemma 1 homogenize all masks in k sharings into one. The proof is as follows.

Proof. We apply a pivot technique to $\vec{Z}''$. Let $\epsilon \in \mathbb{K}$. We notice that the new sharing $\vec{Z}'' = \vec{Z}' + (L'_2\epsilon, \epsilon, 0, \dots, 0)$, also represents the same unmasked value as $\vec{Z}'$ does. Indeed, $\langle L', Z' \rangle = X'$, and $\langle L', (L'_2\epsilon, \epsilon, 0, \dots, 0) \rangle = L'_2\epsilon + L'_2\epsilon = 0$. By choosing $\epsilon = \vec{Z}'_2 + \vec{Z}_2$, we get for $\vec{Z}''$:

$$\vec{Z}'' = (Z'_1 + L'_2(Z'_2 + Z_2), Z_2, Z'_3, \dots, Z'_n).$$

Therefore, $\vec{Z}''$ now has the same the second share (coordinate at position 2) with $\vec{Z}$. The complete homogenization is thus the repetition of this process for all the coordinates $i \in \{2, \dots, n\}$. Notice that this algorithm does leak information neither on $\vec{Z}$ nor on $\vec{Z}'$, since it consists only of additions of masks to a sharing from an independent sharing. It is akin a refresh procedure albeit where the new masks are actually a compensation of $\vec{Z}'$ masks by those of $\vec{Z}$, whilst keeping the masking invariant of Equ. 1. Actually, it is a refresh algorithm using the masks of the difference $\vec{Z} \oplus \vec{Z}'$. □

C IPM-FD with $k = 2$

By using Magma [21], we present some instances for IPM-FD with $k = 2$, in particular $\mathbb{K} = \mathbb{F}_{2^4}$ in Tab. 3 and $\mathbb{K} = \mathbb{F}_2$ in Tab. 4, respectively. Interestingly, we notice that for $\mathbb{K} = \mathbb{F}_2$ the best minimum distance of $\mathbf{H}^{\perp}$ is equal to *BKLC(GF(2), n, 2)*, where n is the same as in the Tab. 4.

Table 3: Examples with $\mathbb{K} = \mathbb{F}_{2^4}$, d_b and d_w are side-channel security orders at bit-level and word-level, respectively.

	Inputs		Outputs of Alg. 4 and Alg. 5		
	#faults d_f	d_w	n	d_b	Setting
IPM	0	0	1	0	$\mathbf{H}^{\perp} = \begin{pmatrix}1\end{pmatrix}$
	0	1	2	2	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & X^5\end{pmatrix}$
	0	2	3	5	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & X^5 & X^{10}\end{pmatrix}$
	0	3	4	7	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & X^5 & X^9 & X^{13}\end{pmatrix}$
	0	4	5	9	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & X^5 & X^9 & X^{12} & X^1\end{pmatrix}$
	0	5	6	11	$BKLC(GF(2), 4 * 6, 4) \simeq [24, 4, 12]$
	0	6	7	13	$BKLC(GF(2), 4 * 7, 4) \simeq [28, 4, 14]$
IPM-FD	1	0	2	0	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 0 \\ 0 & 1\end{pmatrix}$
	1	1	3	2	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 0 & X^5 \\ 0 & 1 & X^{10}\end{pmatrix}$
	1	2	4	4	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 0 & X^5 & X^{11} \\ 0 & 1 & X^{11} & X^4\end{pmatrix}$

Table 4: Examples with $\mathbb{K} = \mathbb{F}_2$, d_w and d_b are security orders at word-level and bit-level, respectively.

	Inputs		Outputs of Alg. 4 and Alg. 5		
	#faults d_f	d_w	n	d_b	Setting
IPM	0	0	1	0	$\mathbf{H}^{\perp} = \begin{pmatrix}1\end{pmatrix}$
	0	1	2	1	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 1\end{pmatrix}$
	0	2	3	2	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 1 & 1\end{pmatrix}$
	0	3	4	3	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 1 & 1 & 1\end{pmatrix}$
	0	4	5	4	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 1 & 1 & 1 & 1\end{pmatrix}$
	0	5	6	5	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1\end{pmatrix}$
	0	6	7	6	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1\end{pmatrix}$
	0	7	8	7	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{pmatrix}$
	0	8	9	8	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{pmatrix}$
	0	9	10	9	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\end{pmatrix}$
IPM-FD	1	0	2	0	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 0 \\ 0 & 1\end{pmatrix}$
	1	1	3	1	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 0 & 1 \\ 0 & 1 & 1\end{pmatrix}$
	1	2	5	2	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1\end{pmatrix}$
	1	3	6	3	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0\end{pmatrix}$
	1	4	8	4	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0\end{pmatrix}$
	1	5	9	5	$\mathbf{H}^{\perp} = \begin{pmatrix}1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1\end{pmatrix}$