Orthogonal Direct Sum Masking

A Smartcard Friendly Computation Paradigm in a Code, with Builtin Protection against Side-Channel and Fault Attacks

Julien BRINGER[1], Claude CARLET[2], Hervé CHABANNE[1,3], Sylvain GUILLEY[3,4] and Houssem MAGHREBI[1].

[1] Morpho, 18 Chaussée Jules César, 95520 Osny, France.
[2] LAGA, UMR 7539, CNRS, Department of Mathematics, University of Paris XIII and University of Paris VIII, 2 rue de la liberté, 93 526 Saint-Denis Cedex, France.
[3] Institut Mines Télécom, Crypto Group, 37/39 rue Dareau, 75 634 Paris Cedex 13, France.
[4] Secure-IC S.A.S., 80 avenue des Buttes de Coësmes, 35 700 Rennes, France.

Abstract. Secure elements, such as smartcards or trusted platform modules (TPMs), must be protected against implementation-level attacks. Those include side-channel and fault injection attacks. We introduce ODSM, Orthogonal Direct Sum Masking, a new computation paradigm that achieves protection against those two kinds of attacks. A large vector space is structured as two supplementary orthogonal subspaces. One subspace (called a code $\mathcal{C}$) is used for the functional computation, while the second subspace carries random numbers. As the random numbers are entangled with the sensitive data, ODSM ensures a protection against (monovariate) side-channel attacks. The random numbers can be checked either occasionally, or globally, thereby ensuring a fine or coarse detection capability. The security level can be formally detailed: it is proved that monovariate side-channel attacks of order up to $d_{\mathcal{C}} - 1$, where $d_{\mathcal{C}}$ is the minimal distance of $\mathcal{C}$, are impossible, and that any fault of Hamming weight strictly less than $d_{\mathcal{C}}$ is detected. A complete instantiation of ODSM is given for AES. In this case, all monovariate side-channel attacks of order strictly less than 5 are impossible, and all fault injections perturbing strictly less than 5 bits are detected.

Keywords: Masking countermeasure, trans-masking, fault detection, orthogonal supplementary spaces, linear codes, minimal and dual distances, AES

1 Introduction

Side-channel analysis (SCA) and fault analysis (FA) are nowadays well known and most designers of secure embedded systems are aware of them. Since the first public reporting of these threats in 1996, a lot of effort has been devoted

towards the research about these attacks and the development of corresponding protections. Several countermeasures have been proposed, but usually tackling only side-channel analysis [16] or (exclusively) fault analysis [15].

Masking is one of the most efficient countermeasures to thwart SCA. The most critical part when applying masking to secure cryptographic implementations, is to protect their non-linear operations (e.g., the *substitution boxes*, S-boxes). Commonly, there are three strategies [20]: the *global look-up table* (GLUT), the *table re-computation method*, and the *S-box secure calculation*. The GLUT method seems to be the most appropriate method: its timing performances are ideal since it requires only one memory transfer. However, the GLUT method has an exponential increase of the table size with the amount of entropy (e.g., the number of masks used).

A recent line of works known as Low-Entropy Masking Schemes (LEMS) has investigated possibilities to preserve the security level of masked implementations with reduced randomness requirements [3]. In fact, the mask is generated within a subset of all possible values. For instance, the set of masks could be a set of codewords, to reduce the overhead in terms of computational resources and entropy. Therefore, the LEMS scheme is still compatible with a table recomputation method, and a representative computation is sketched below:

- One random number d is drawn.
- (Optionally: tables are recomputed with d as a mask).
- User data (e.g., plaintext) is masked.
- Computations are done within such masked representation.
- The result (e.g., ciphertext) is demasked.

This is obviously only a first-order masking scheme, because it is still possible to combine two leaks resulting from a reuse of the mask, for instance the information leaked during the table recomputation and then during the computation. But, it can also be made more complicated by using shuffling [27] (especially during the table recomputation).

Related works. The GLUT and table recomputation masking schemes have been described several times in the case of AES [17,12,5]. However, those countermeasures stick to the word size k (e.g., $k = 8$ for AES), possibly with multiple shares of size k. In this paper, we propose a new LEMS scheme, called *Orthogonal Direct Sum Masking* (ODSM). Like the wire-tap masking [7], it can work with any amount of added entropy (not necessarily by increments of k bits). Compared to wire-tap masking, ODSM takes advantage of an orthogonal projection to ease operations in a linear code of length $n > k$ and dimension k.

Contrasted to the state-of-the-art masking and fault protection, our masking scheme presents many innovative features. In fact, using one share, ODSM ensures a practical security against monovariate SCA (but still high-order) and provides the possibility of removing the mask without the knowledge of it. Moreover, an overwhelming advantage of this new scheme over any other masking technique, is the capability to detect some injected faults while the main goal is

to ensure security against SCA. Indeed, such synergy between SCA and FA protections does not exist for other masking schemes [6]. Nonetheless, we note that *dual-rail logic* too enjoys the simultaneous protection against SCA and FA [2].

Eventually, for algorithms with large S-boxes, like AES ($k = 8$, to be compared with $k = 4$ for PRESENT), we show how to switch from ODSM to the classical first-order perfect masking using table recomputations.

Contributions. We introduce a masking scheme provably secure against monovariate attacks that uses a customizable entropy (namely $n - k$ bits, choice of the designer). With respect to the state-of-the-art, an encoding function mixes optimally the randomness with the sensitive data, in order to achieve the best protection against side-channel attacks. Additionally, the $n - k$ redundant bits injected in the computation can be leveraged to check for the injection of errors. We show that the minimal distance $d_{\mathcal{C}}$ of the $[n, k, d_{\mathcal{C}}]$ code determines the minimal weight of errors to be injected for (possibly) bypassing the error sanity checks of the ODSM scheme. We also apply the ODSM to the AES, with full details on the way the computations in the linear codes are performed.

Outline. The rest of the article is organized as follows. The theory about the linear algebra and codes, and how it is applied in the ODSM countermeasure, are the topic of Sec. 2. The practical implementation of the ODSM scheme for the AES cipher and possible improvements are given in Sec. 3. Section 4 provides some discussions about the advance on the field of implementation security conveyed by ODSM, and especially a comparative analysis with other schemes, plus a presentation of ODSM distinguishing features. Conclusions and perspectives are in Sec. 5. Technical results and examples are relegated in the appendices A and B.

2 Theoretical foundations

In this section, we first recall in Sec. 2.1 the basic notions of linear algebra and linear codes needed to describe our masking scheme. We intentionally skip the proofs of well-known propositions, but provide proofs for non classical results. Then, Sec. 2.2 contains the generic description of the ODSM; the construction is fully defined by a linear code $\mathcal{C}$ (with specific properties). The ODSM scheme is briefed in Sec. 2.3. The security attributes of this masking scheme are given in Sec. 2.4, thanks to properties of the code $\mathcal{C}$.

2.1 Basic notions of linear algebra and linear codes

Let k and n be two integers, such that $k \leq n$. The set of n-bit vectors, noted $\mathbb{F}_2^n$, is endowed with a structure of space vector. Let $\mathcal{C}$ be a subspace of $\mathbb{F}_2^n$ of dimension k. Then, $\mathcal{C}$ is a linear code of length n and dimension k.

Definition 1 (supplement of a space vector). *$\mathcal{C}$ can be completed with some vectors in order to spawn $\mathbb{F}_2^n$. Those vectors define the supplement $\mathcal{D}$ of $\mathcal{C}$ in $\mathbb{F}_2^n$. We write $\mathbb{F}_2^n = \mathcal{C} \oplus \mathcal{D}$ to say that $\mathbb{F}_2^n$ is the direct sum of $\mathcal{C}$ and $\mathcal{D}$.*

Remark 1. We note that the same symbol "$\oplus$" is used for the direct sum and for the addition of vectors, which is uncommon but which does not create any ambiguity.

A linear code is spawned by a basis: the matrix whose rows consist in the basis vectors is called a *generating matrix*. We denote by G (resp. H) the generating matrix of $\mathcal{C}$ (resp. $\mathcal{D}$, the supplement of $\mathcal{C}$). Then, we have that every element $z \in \mathbb{F}_2^n$ can be written uniquely as:

$$z = c \oplus d \ , \tag{1}$$

where $c \in \mathcal{C}$ and $d \in \mathcal{D}$. Now, as all $c \in \mathcal{C}$ (resp. $d \in \mathcal{D}$) can also be written uniquely as xG (resp. yH), for a given $x \in \mathbb{F}_2^k$ (resp. $y \in \mathbb{F}_2^{n-k}$), we have the following equation:

$$z = xG \oplus yH \ . \tag{2}$$

In the following definitions, we formalize the notions of minimal and dual distance of a linear code.

Definition 2 (minimal distance). *The minimal distance $d_{\mathcal{C}}$ of a linear code $\mathcal{C}$ of length n and dimension k is the minimal Hamming distance of any two different elements of $\mathcal{C}$. We say that $\mathcal{C}$ has parameters $[n, k, d_{\mathcal{C}}]$.*

Definition 3 (dual distance). *The dual distance $d_{\mathcal{C}}^{\perp}$ of a code $\mathcal{C}$ is the minimal Hamming weight $w_H(z)$ of a nonzero vector $z \in \mathbb{F}_2^n$ such as $\sum_{c \in \mathcal{C}} (-1)^{z \cdot c} \neq 0$, where $z \cdot c$ is the scalar product between z and c ($z \cdot c = \sum_{i=1}^{n} z_i c_i$, or equivalently $z \cdot c = zc^{\mathsf{T}} \in \mathbb{F}_2$ using matrix notations). We recall that the Hamming weight $w_H(z)$ of a vector z is $w_H(z) = \sum_{i=1}^{n} z_i$.*

Definition 4 (orthogonal). *The orthogonal of a set $\mathcal{C} \subseteq \mathbb{F}_2^n$ is the space vector $\mathcal{C}^{\perp}$ defined as $\{d \in \mathbb{F}_2^n | \forall c \in \mathcal{C}, d \cdot c = 0\}$. When $\mathcal{C}$ is a linear code, $\mathcal{C}^{\perp}$ is called the dual code of $\mathcal{C}$. The generating matrix of $\mathcal{C}^{\perp}$ is called the parity matrix of $\mathcal{C}$.*

Proposition 1. *For a linear code $\mathcal{C}$, $d_{\mathcal{C}}^{\perp} = d_{\mathcal{C}^{\perp}}$.*

Indeed, in linear algebra, the dual code $\mathcal{C}^{\perp}$ can be seen as the *kernel* (or *null space*) of the code $\mathcal{C}$. We have the following Theorem for a linear code $\mathcal{C}$.

Theorem 1 (rank-nullity). $\dim(\mathcal{C}) + \dim(\mathcal{C}^{\perp}) = \dim(\mathbb{F}_2^n) = n$, *where* $\dim(\cdot)$ *is the dimension of the vector space.*

As a direct consequence of Theorem 1, we have $\dim(\mathcal{C}^{\perp}) = n - k$. However, $\mathcal{C}$ and $\mathcal{C}^{\perp}$ are not necessarily supplementary, i.e., we do not have $\mathcal{C} \cap \mathcal{C}^{\perp} = \{0\}$. For instance, if $\mathcal{C}$ is autodual, then $\mathcal{C} = \mathcal{C}^{\perp}$.

In the following Proposition 2, we exhibit a necessary and sufficient condition to have $\mathcal{C}$ and $\mathcal{C}^{\perp}$ supplementary in $\mathbb{F}_2^n$.

Proposition 2 (Condition for $\mathbb{F}_2^n = \mathcal{C} \oplus \mathcal{C}^\perp$). *Without loss of generality (a permutation of coordinates might be necessary), we can assume that the generating matrix of $\mathcal{C}$ is systematic, and thus takes the form $[I_k \| M]$, where I_k is the $k \times k$ identity matrix. The supplement $\mathcal{D}$ of $\mathcal{C}$ is equal to $\mathcal{C}^\perp$ if and only if (iff) the matrix $I_k \oplus MM^\mathsf{T}$ is invertible.*

Proof. Because of the dimensions, the supplement of $\mathcal{C}$ (named $\mathcal{D}$) is equal to $\mathcal{C}^\perp$ if and only if $\mathcal{C} \cap \mathcal{C}^\perp = \{0\}$. In the systematic form, $\mathcal{C}$ has the generating matrix $[I_k \| M]$ and parity matrix $[M^\mathsf{T} \| I_{n-k}]$. So, the condition is that: $\forall (x, y) \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$, the system of two equations $x = yM^\mathsf{T}$ and $xM = y$ has only $(0, 0)$ as solution. This is equivalent to saying that the equation $x = xMM^\mathsf{T}$ has only the trivial solution, and thus that the matrix $I_k \oplus MM^\mathsf{T}$ is invertible. □

When $\mathcal{D} = \mathcal{C}^\perp$, there is an orthogonal projection. Indeed, we thus have $GH^\mathsf{T} = 0$ (the all-zero $k \times (n-k)$ matrix). In this case, H is the *parity matrix* of code $\mathcal{C}$. So, in Eq. (2), x and y can be recovered from z, as follows:

$$x = zG^\mathsf{T}(GG^\mathsf{T})^{-1} \ , \tag{3}$$

$$y = zH^\mathsf{T}(HH^\mathsf{T})^{-1} \ . \tag{4}$$

Notice that given that G is a basis for $\mathcal{C}$, it is composed of linearly independent vectors of $\mathbb{F}_2^n$. Hence GG^T is a $k \times k$ invertible matrix. Similarly, provided H is a basis for $\mathcal{D}$, HH^T is a $(n-k) \times (n-k)$ invertible matrix.

It follows from Eq. (1) that the projection $P_\mathcal{C}$ (resp. $P_\mathcal{D}$) of $z \in \mathbb{F}_2^n$ on $\mathcal{C}$ (resp. $\mathcal{D}$) is given by:

$$P_\mathcal{C} : \mathbb{F}_2^n \to \mathcal{C}, \quad z \mapsto c = zG^\mathsf{T}(GG^\mathsf{T})^{-1}G \ , \tag{5}$$

$$P_\mathcal{D} : \mathbb{F}_2^n \to \mathcal{D}, \quad z \mapsto d = zH^\mathsf{T}(HH^\mathsf{T})^{-1}H \ . \tag{6}$$

2.2 Definition of the masking scheme

Data representation. For the masking scheme, we choose $\mathcal{C}$ and $\mathcal{D}$ such as $\mathcal{D} = \mathcal{C}^\perp$. Using the property of Eq. (1), we suggest to represent any vector z of $\mathbb{F}_2^n$ as the sum of two codewords $z = c \oplus d$. The coded sensitive data is $c \in \mathcal{C}$, while the mask is $d \in \mathcal{D}$.

So, to protect a k bit sensitive data x, $(n-k)$ random bits are required. Those are denoted by y; the mask is equal to $d = yH$. The idea is that the information are codewords, and that the masks act as intentionally added noise. But, as the information and the noise live in two supplementary subspaces, it is always possible to recover both, using Eq. (3) and Eq. (4).

Computation. The goal is to carry out the computation within the orthogonal direct sum representation of Eq. (2). We assume that all the steps in computations can be represented as $\mathbb{F}_2^k \to \mathbb{F}_2^k$ functions. For instance, this is indeed the case for AES [19]. This block cipher manipulates bytes ($k = 8$). Even operations that operate on larger structures, such as MixColumns ($\mathbb{F}_2^{32} \to \mathbb{F}_2^{32}$),

can be decomposed as operations on bytes, by using `xtime` [19, Sec. 4.2.1], for instance. In the sequel, we simply denote by *word* a k-bit word (and precise "n-bit" otherwise).

In this section, we give mathematical definitions of the ODSM scheme; examples are provided in Sec. 3.

We make the difference between three different operations:

(i) two-operand operations, that are usually exclusive-or operations, between two words;
(ii) linear transformations of one word, referred to as $\mathcal{L}$ (of matrix L);
(iii) non-linear transformations of one word, referred to as S (like the "S" of an S-box).

Computation of type (i). The exclusive-or in $\mathbb{F}^k$ is a straightforward operation to port after encoding in $\mathbb{F}_2^n$, because it remains *the same* in $\mathbb{F}_2^n$ (it is the *additive law* in both space vectors). For instance, the key addition step is as follows: let $z_1 = x_1 G \oplus y_1 H$ be the coded and masked element of $\mathbb{F}_2^n$ used in ODSM to represent the plaintext byte x_1, and k the key. Then, the secure key addition is:

$$z_2 = z_1 \oplus kG = (x_1 \oplus k)G \oplus y_1 H \ . \tag{7}$$

Computation of type (ii). Any linear operation $\mathcal{L} : \mathbb{F}_2^k \to \mathbb{F}_2^k$ can be turned into a masked operation, where the mask is unchanged by choice. This masked linear operation is denoted by $\mathcal{L}' : \mathbb{F}_2^n \to \mathbb{F}_2^n$. We call L and L' the matrices of the linear operations (i.e., $\mathcal{L}(x) = xL$ and $\mathcal{L}'(z) = zL'$). The matrix L' is defined from L by:

$$L' = G^{\mathsf{T}} \left(G \cdot G^{\mathsf{T}}\right)^{-1} LG \oplus H^{\mathsf{T}} \left(H \cdot H^{\mathsf{T}}\right)^{-1} H \ . \tag{8}$$

Indeed, let $z = xG \oplus yH$, one has:

$$\begin{aligned} zL' &= \underbrace{\left(zG^{\mathsf{T}} \left(G \cdot G^{\mathsf{T}}\right)^{-1}\right)}_{x} LG \oplus \underbrace{\left(zH^{\mathsf{T}} \left(H \cdot H^{\mathsf{T}}\right)^{-1}\right)}_{y} H \\ &= (xL)G \oplus yH \ . \end{aligned} \tag{9}$$

Thus, the linear operation consists in a product of the n-bit word by an $n \times n$ matrix. Even if this matrix is stored "uncompressed" in memory, it consists only in n words of n bits (whereas an S-box would require 2^n words of n bits).

Computation of type (iii). Non-linear operations $S : \mathbb{F}_2^k \to \mathbb{F}_2^k$ are simply recomputed. As for the case of linear functions, we denote by $S' : \mathbb{F}_2^n \to \mathbb{F}_2^n$ the masked non-linear operation. It is computed (off-line, once for all) as:

$$\forall z \in \mathbb{F}_2^n, \quad S'(z) = S(zG^{\mathsf{T}}(GG^{\mathsf{T}})^{-1})G \oplus zH^{\mathsf{T}}(HH^{\mathsf{T}})^{-1}H \ , \tag{10}$$

where we also make the assumption that the mask value is unchanged when traversing S'. Indeed, let us write $z = xG \oplus yH$. Thus $S'(z) = S(x)G \oplus yH$.

2.3 Orthogonal direct sum masking (ODSM)

The purpose of this short subsection is to recapitulate the principle of ODSM.

From a side-channel analysis perspective, it is a masking scheme that belongs to the class of:

- *Boolean additive* masking schemes (i.e., the mask is inserted with + in $\mathbb{F}_2^n$)
- Global Look-Up Table (GLUT), compatible with the table recomputation scheme [20] (see how to practically switch between schemes at Sec. 3.3)
- Low-entropy masking schemes [29] (the injected entropy is an integer $n-k$ that can be equal to 1, 2, etc.)

Every new encryption unfolds as presented in Alg. 1, where the state is denoted by z. The steps in gray represent optimizations in memory size (they are optional, because fault detection is disabled when they are implemented).

Algorithm 1 Big picture for the secure computation of a block cipher using ODSM

1: Draw a random variable y uniformly distributed in $\mathbb{F}_2^{n-k}$
2: (Optionally: precompute the masked non-linear tables $S'_{\text{recomp}} : \mathbb{F}_2^k \to \mathbb{F}_2^k$ from the genuine S-box $S : \mathbb{F}_2^k \to \mathbb{F}_2^k$ as per Eq. (13) — this aspect will be detailed in Sec. 3.3, since it is an optimization)
3: Encode and mask the plaintext x, as $z = xG \oplus yH$ (refer to Eq. (2))
4: Schedule the key (its protection is out of the scope of this paper; Indeed, the key is considered as non-sensitive, from a SCA standpoint — see discussion in [22, §4])
5: Iterate, for each operation of the block cipher:
   (i) if it is a key addition, multiply the key word by G and add it to z, as per Eq. (7)
   (ii) if it is a *linear* operation, apply L' to z as per Eq. (8)
   (iii) if it is a *non-linear* operation, apply S' to z as per Eq. (10)
   (or apply S'_{recomp} through procedure depicted in Alg. 4 if this option is selected)
6: Whenever required, use the relation Eq. (6) to verify that the mask has not been altered (in case of a fault attack)
7: At the end of the computation, use the relation Eq. (3) to obtain the unmasked ciphertext

2.4 Security properties

Security against SCA. We recall that a *sensitive variable* depends on the plaintext and on the key. Thus, neither the plaintext (nor the ciphertext) nor the key (master or round keys) are sensitive (see e.g., [22, §4]). This means that we consider only vertical attacks, where the attacker needs to collect a sufficient amount of traces to recover the secret.

It can be seen from subsection 2.3 that ODSM has been designed such that all sensitive variables are masked (by $d = yH$), as in Eq. (2). The verification

against fault attacks (line 6 of Alg. 1) is non-sensitive, because it leaks the mask, which is not sensitive *alone* (we also assume that it cannot be recovered in one trace, i.e., horizontal attacks [24] do not apply). Such masking scheme has been proved perfectly masked (against monovariate attacks) by Blömer et al. in [5].

Now, if the device is leaking at order one (i.e., there is no "glitch" nor "cross-coupling"), then, the order of resistance of the ODSM scheme is quantified by Theorem 2, whose extensive proof is given in Appendix A.

Theorem 2 (Order of resistance). *ODSM can be attacked by monovariate high-order SCA only at order* $j \geq d_{\mathcal{C}}$.

Remark 2. ODSM enjoys only a security against monovariate attacks (i.e., combining two leakage samples exhibits a dependency with the sensitive variable). However, let us mention this is the state-of-the-art of practically implementable masking schemes. Provably secure second-order masking schemes (e.g., [21,22,11]) are admittedly complex to be implemented in practice, owing to their long execution time. Besides, bi-variate attacks can be made very challenging by coupling the masking scheme with *shuffling*. Indeed, in block ciphers such as the AES,

- byte-oriented operations (AddRoundKey, SubBytes, ShiftRows) can be conducted in whatever order (16 possibilities per byte),
- column-oriented operations (MixColumns – unless the implementation with `xtime` is used, in which case this is also a byte-oriented operation) can be conducted in whatever order (amongst the 4! possible orders).

Security against fault injection attacks. In Alg. 1, the state z, throughout the computation, is masked by the same quantity yH, for a $y \in \mathbb{F}_2^{n-k}$ chosen randomly at the beginning of the computation. So, the value of the mask can be checked from times to times (as indicated in line 6 of Alg. 1). The verification takes the following form:

$$P_{\mathcal{D}}(z) \stackrel{?}{=} yH \ . \tag{11}$$

This operation is sibling to the computation of a *syndrome*.

Let us analyse the exact conditions for the detection to work. We consider a perturbation as the addition to the state z of a random error ε ($z \leftarrow z \oplus \varepsilon$). Like z (recall $z = xG \oplus yH$, see Eq. (2)), the fault can be uniquely written as:

$$\varepsilon = eG \oplus fH \ , \text{ where } e \in \mathbb{F}_2^k \text{ and } f \in \mathbb{F}_2^{n-k} \ . \tag{12}$$

The fault is undetected if $P_{\mathcal{D}}(z \oplus \varepsilon) = (y \oplus f)H = yH$, which is equivalent to have $f = 0$.

If the faults ε are uniformly distributed over $\mathbb{F}_2^n$, then the probability of non-detection is $2^{-(n-k)}$. This probability can be regarded as high. Indeed, it is known that very few faults (sometimes only one or two [23]) can expose the complete key of an AES. However, there are two reasons for the fault injections on ODSM to be more difficult to carry out in practice.

First of all, multiple checks can be done during the algorithm, without any overhead (apart the test of Eq. (11)), because the mask yH is, by design, an invariant throughout the computation.

Second, the undetected faults are indeed very special and most probably difficult to produce in practice. Indeed, $f = 0$ means that $\varepsilon \in \mathcal{C}$ (recall Eq. (12)). Now, we assumed *conservatively* that faults were *uniformly* distributed. Experimentally, it is rather easier to produce faults that have a low Hamming weight. Indeed, when setting up the perturbation source, the stress is first applied gently, and then increased until some effect becomes observable. This approach allows to avoid the activation of sensors due to too heavy a stress, and also to avoid the circuit simply from crashing due to excessive malfunctions. So, for instance considering *overclocking*, the clock frequency is gradually increased, until the first error appears [1]. As a matter of fact, there will be initially only one fault on the *critical path*, thereby causing only 1 bit-flip. This means that easy to inject faults have low Hamming weight. But as ε must be in $\mathcal{C}$ (and nonzero) to have an effect while being undetected, it must have a Hamming weight of at least $d_{\mathcal{C}}$. The likelihood of such faults is probably setup-dependent, but is "informally speaking" much smaller than the announced probability of $2^{-(n-k)}$.

3 Implementation of AES following the ODSM scheme

3.1 Example with a binary linear code $\mathcal{C}$ of parameters $[16, 8, 5]$

Most smartcards and TPMs are still byte-oriented. In this section, we present the case $k = 8$ and $n = 16$, suitable for the AES. Indeed, choosing $n < 16$ would result in ignoring some bits in the processor registers and memory words. At the opposite, if a hardware target (ASIC or FPGA) had been chosen, any value of n would have been eligible, thereby allowing for finer security / overhead tradeoffs.

The problem is to find a code of length n ($n \leq 16$) and of dimension $k = 8$, with minimal distance as large as possible, such as its dual is its supplementary. It happens that there exists a linear code with the expected properties and good parameters: the code of parameters $[16, 8, 5]$ (see generator matrix G in Appendix B) has a supplementary dual, and minimal distance $d_{\mathcal{C}} = 5$ (which is maximal for a linear code). Any linear code of length $n < 16$ and dimension $k = 8$ has a minimal distance strictly smaller than 5. As the $[16, 8, 5]$ code is very attractive, we use it as an example in this paper.

Remark 3. By Theorem 2, using this linear code, we protect the AES against all monovariate high-order attacks of order $j \leq 4$. Moreover, all fault injections perturbing 4 bits or less are detected.

3.2 Efficient implementation of linear functions

First of all, we notice that matrices described in Sec. 2 are precomputed. As for AES, the matrices for G and H (See Eq. (16)), and for $G^{\mathsf{T}} \cdot (G \cdot G^{\mathsf{T}})^{-1}$ and

$H^\mathsf{T} \cdot (H \cdot H^\mathsf{T})^{-1}$ (Eq. (17)) are precomputed. Also, the one non-trivial linear operation, namely `xtime` (Eq. (18)) is also stored masked, as L' (Eq. (19)).

Besides, the computation of vector–matrix products, a priori of $n \times n$ complexity, can be enhanced by using the natural parallel feature of processors. For instance, one can compute a vector–matrix product in a *bitslice* manner [4]. The algorithm for the computation of vector–matrices products is given in Alg. 2. Moreover, a version in `C` language is in Alg. 3.

Algorithm 2 Vector–matrix product (bitslice approach on $k = 8$ bits)

Input: $v \in \mathbb{F}_2^8$ and $M \in (\mathbb{F}_2^8)^8$, whose rows are denoted by $r[i] \in \mathbb{F}_2^8$ (for $1 \leq i \leq 8$)
Output: $w = vM \in \mathbb{F}_2^8$

1: $w \leftarrow 0$ (a vector of 8 bits [i.e., a *line*, as opposed to a *column*])
2: **for** $i \in [\![1, 8]\!]$ **do**
3: $\quad w \leftarrow w \oplus v[i] \wedge r[i]$ ▷ Interpret the AND ($\wedge$) as: $\begin{cases} 0 \wedge r[i] = 0 \text{ and} \\ 1 \wedge r[i] = r[i] \end{cases}$
4: **end for**
5: **return** w

Algorithm 3 `C` code, corresponding to Alg. 2

```
#include <stdint.h>
uint8_t w=0;
for( unsigned i=0; i<8; ++i )
        w ^= ( v >> i )&1 ? r[i] : 0;
return w;
```

The vector–matrix products are processed by blocks of 8×8. We notice that G and H can be written in systematic form, as in Eq. (16). Thus, as the first (resp. last) block of G (resp. H) is I_8, no computation is involved. Moreover, G and H take only 64 bits of ROM each.

3.3 Efficient implementation of non-linear functions

The GLUT approach presented in Sec. 2 has the advantage of being efficient, but even for small n, it is costly in memory size. For instance, for AES, the GLUT size in memory is $n2^n$ bits (see Eq. (10)). Therefore, the table recomputation approach would be welcome. It happens that ODSM has the nice property to support both approaches.

A mask, or a pair of masks $(x', x'') \in \mathbb{F}_2^k \times \mathbb{F}_2^k$, for the S-box recomputation is chosen randomly. It can very well be that $x' = x''$ without jeopardizing the monovariate security against side-channel attacks. Then, we compute for all

$x \in \mathbb{F}_2^k$:

$$S'_{\text{recomp}}(x) = S(x \oplus x') \oplus x'' \ . \tag{13}$$

The "trans-masking" operation (for switching between ODSM and precomputed tables) is described in Alg. 4. It is a rare example of *straightforward* switch between two masking schemes (see [13,26] for other examples).

Algorithm 4 Masked application of an S-box on z with a switching between ODSM and a precomputed table. Input: $z = xG \oplus yH$ / Output: $z' = S(x)G \oplus yH$

1: $z \leftarrow z \oplus x'G$,	[Masked with x' in $\mathcal{C}$ and d in $\mathcal{D}$]
2: $x \leftarrow zG^{\mathsf{T}}(GG^{\mathsf{T}})^{-1}$,	[Perfect masking with x' in $\mathbb{F}_2^k$]
3: $x \leftarrow S'_{\text{recomp}}(x)$,	[Secure masked look-up]
4: $z' \leftarrow xG \oplus yH$,	[Remasking with $d = yH$ in $\mathcal{D}$]
5: $z' \leftarrow z' \oplus x''G$.	[Demasking x'' in $\mathcal{C}$]

Nonetheless, we stress two drawbacks of the table recomputation approach:

1. It incurs a time penalty for the recomputation preliminary stage (Eq. (13)).
2. Fault detection is not possible during the evaluation of the precomputed table, because the $(n-k)$ redundant bits are no longer used (the computation falls back on k bits). Still, the security against monovariate side-channel analysis is granted, since the sensitive variable is manipulated *perfectly masked* with (x', x''). So, for AES, we recommend to do the check $P_{\mathcal{D}}(z) \stackrel{?}{=} yH$ before all look-ups in a precomputed table.

4 Discussion

Remark 4. We highlight in this remark the difference with coding xG in an *Additive White Gaussian Noise* (AWGN) channel, in the context of *digital coding theory*. Our ODSM scheme shares with error detection coding that the errors of low weight can be detected (see analysis in subsection 2.4). But in addition, it manages to handle data in the presence of intentional "noise" of large Hamming weight, namely all the nonzero vectors of H. The wire-tap coding [7] shares the same features.

Remark 5. In the masking with several shares (e.g., Goubin and Patarin [14]), the mask changes throughout the implementation. Indeed, for instance, with two shares Z_1 and Z_2, a linear function $\mathcal{L}$ is applied by calling $\mathcal{L}$ on each share. As a matter of fact, if initially the sensitive variable is $Z = Z_1 \oplus Z_2$, the value of $\mathcal{L}(Z)$ is indeed shared as $\mathcal{L}(Z_1)$ on the one hand, and $\mathcal{L}(Z_2)$ on the one hand. Therefore, after demasking, the exclusive-or of $\mathcal{L}(Z_1)$ and $\mathcal{L}(Z_2)$ yields $\mathcal{L}(X)$. But if Z_2 is a random mask, then it takes value $\mathcal{L}(Z_2)$ after the function $\mathcal{L}$. In contrast, in ODSM, the mask is constrained to be untouched during the whole

computation. This makes verifications much more convenient since the same verification can be done irrespective to the place in the cryptographic algorithm (said differently, the verification is not contextual).

Remark 6. When $n = k+1$, the ODSM countermeasure is equivalent to the low-entropy masking scheme proposed in [3]. It consists in having only two vectors in $\mathcal{D}$, namely $(0000)_2$ and $(1111)_2$ (the scheme is applied to the nibble-oriented PRESENT, i.e., $k = 4$). It is shown in [3] to resist first-order SCA (theoretically and by laboratory experiments).

5 Conclusions and Perspectives

The ODSM masking scheme has two *security distinctive features* over other countermeasures against SCA and FA:

1. It resists monovariate attacks of degree $d_{\mathcal{C}} - 1$ ($d_{\mathcal{C}} \geq 1$ if $\mathcal{C}$ is simple and non-empty, but in practice $d_{\mathcal{C}} \gg 1$, e.g. $d_{\mathcal{C}} = 5$ when $\mathcal{C}$ is a $[16, 8, 5]$ code).
2. It can detect faults with probability $1 - 2^{-(n-k)}$ assuming the attacker is able to inject faults uniformly in $\mathbb{F}_2^n$; However, in practice, undetected faults ε must meet a strong criteria, namely $\varepsilon \in \mathcal{C}$, which implies in particular that $w_H(\varepsilon) \geq d_{\mathcal{C}}$, which is for instance 5 for the $[16, 8, 5]$ code $\mathcal{C}$.

Both properties result from the fact the computation in ODSM is carried out, from end to end, in a coset $\mathcal{C} \oplus d$ of the linear code $\mathcal{C}$, where $d \in \mathcal{D} = \mathcal{C}^{\perp}$ is a random mask chosen before every new encryption. The initial randomness of d allows for the protection against monovariate and vertical SCA, whereas the constantness of d throughout the encryption allows for episodic checks against FA. The adaptation of the $[16, 8, 5]$ solution to other *form factors* (i.e., different values of k & n) raises the interesting problem of finding codes with orthogonal supplementary and large minimal distance. In case such codes do not exist, the *orthogonal* protection could be advantageously be replaced by an *oblique* projection.

Acknowledgments

This work has been partially funded by the ANR project E-MATA HARI. The authors would like to thank Shivam Bhasin, Nicolas Bruneau and Zakaria Najm for insights in the demonstration of Theorem 2 provided in Appendix A. Secure-IC and Télécom-ParisTech are founding members, with DOREMI, of the "Secure Compression Lab". Morpho and Télécom-ParisTech are founders of the "Identity & Security Alliance".

References

1. Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. When Clocks Fail: On Critical Paths and Clock Faults. In *CARDIS*, volume

6035 of *Lecture Notes in Computer Science*, pages 182–193. Springer, April 14-16 2010. Passau, Germany.

2. Shivam Bhasin, Jean-Luc Danger, Florent Flament, Tarik Graba, Sylvain Guilley, Yves Mathieu, Maxime Nassar, Laurent Sauvage, and Nidhal Selmane. Combined SCA and DFA Countermeasures Integrable in a FPGA Design Flow. In *ReConFig*, pages 213–218. IEEE Computer Society, December 9–11 2009. Cancún, Quintana Roo, México, DOI: 10.1109/ReConFig.2009.50, `http://hal.archives-ouvertes.fr/hal-00411843/en/`.
3. Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. A Low-Entropy First-Degree Secure Provable Masking Scheme for Resource-Constrained Devices. In *Proceedings of the Workshop on Embedded Systems Security*, WESS '13, pages 7:1–7:10, New York, NY, USA, September 29 2013. ACM. Montreal, Quebec, Canada. DOI: 10.1145/2527317.2527324.
4. Eli Biham. A Fast New DES Implementation in Software. In Eli Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.
5. Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably Secure Masking of AES. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004.
6. Arnaud Boscher and Helena Handschuh. Masking Does Not Protect Against Differential Fault Attacks. In *FDTC, 5th Workshop on Fault Detection and Tolerance in Cryptography, IEEE-CS*, pages 35–40, aug 2008. DOI: 10.1109/FDTC.2008.12, Washington, DC, USA.
7. Julien Bringer, Hervé Chabanne, and Thanh Ha Le. Protecting AES against side-channel analysis using wire-tap codes. *J. Cryptographic Engineering*, 2(2):129–141, 2012.
8. Claude Carlet. Boolean Functions for Cryptography and Error Correcting Codes: Chapter of the monography Boolean Models and Methods in Mathematics, Computer Science, and Engineering. pages 257–397. Cambridge University Press, Y. Crama and P. Hammer eds, 2010. Preliminary version available at `http://www.math.univ-paris13.fr/~carlet/chap-fcts-Bool-corr.pdf`.
9. Claude Carlet, Jean-Luc Danger, Sylvain Guilley, Houssem Maghrebi, and Emmanuel Prouff. Achieving side-channel high-order correlation immunity with Leakage Squeezing. *Journal of Cryptographic Engineering*, pages 1–15, 2014. DOI: 10.1007/s13389-013-0067-1.
10. Claude Carlet and Philippe Guillot. A New Representation of Boolean Functions. In Marc P. C. Fossorier, Hideki Imai, Shu Lin, and Alain Poli, editors, *AAECC*, volume 1719 of *Lecture Notes in Computer Science*, pages 94–103. Springer, 1999.
11. Jean-Sébastien Coron. Higher Order Masking of Look-up Tables. Cryptology ePrint Archive, Report 2013/700, 2013. `http://eprint.iacr.org/`.
12. Jean-Sébastien Coron and Louis Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 231–237. Springer, August 17-18 2000. Worcester, MA, USA.
13. Blandine Debraize. Efficient and provable Secure Methods for Switching from Arithmetic to Boolean Masking. In *CHES*, September 9-12 2012. Leuven, Belgium.
14. Louis Goubin and Jacques Patarin. DES and Differential Power Analysis. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, volume 1717 of *LNCS*, pages 158–172. Springer, 1999.

15. Marc Joye and Michael Tunstall. *Fault Analysis in Cryptography*. Springer LNCS, March 2011. `http://joye.site88.net/FAbook.html`. DOI: 10.1007/978-3-642-29656-7 ; ISBN 978-3-642-29655-0.

16. 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/`.

17. Thomas S. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In *Fast Software Encryption'00*, pages 150–164. Springer-Verlag, April 2000. New York.

18. Amir Moradi. Statistical tools flavor side-channel collision attacks. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 428–445. Springer, 2012.

19. NIST/ITL/CSD. Advanced Encryption Standard (AES). FIPS PUB 197, Nov 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

20. Emmanuel Prouff and Matthieu Rivain. A Generic Method for Secure SBox Implementation. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, *WISA*, volume 4867 of *Lecture Notes in Computer Science*, pages 227–244. Springer, 2007.

21. Matthieu Rivain, Emmanuelle Dottax, and Emmanuel Prouff. Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. *Fast Software Encryption FSE 2008*, 5086:127–143, 2008.

22. 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.

23. Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault. In Claudio Agostino Ardagna and Jianying Zhou, editors, *WISTP*, volume 6633 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2011.

24. Michael Tunstall, Carolyn Whitnall, and Elisabeth Oswald. Masking Tables - An Underestimated Security Risk. *IACR Cryptology ePrint Archive*, 2013:735, 2013.

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

26. Praveen Kumar Vadnala and Johann Großschädl. Algorithms for Switching between Boolean and Arithmetic Masking of Second Order. In Benedikt Gierlichs, Sylvain Guilley, and Debdeep Mukhopadhyay, editors, *SPACE*, volume 8204 of *Lecture Notes in Computer Science*, pages 95–110. Springer, 2013.

27. Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, 2012.

28. Jason Waddle and David Wagner. Towards Efficient Second-Order Power Analysis. In *CHES*, volume 3156 of *LNCS*, pages 1–15. Springer, 2004. Cambridge, MA, USA.

29. Xin Ye and Thomas Eisenbarth. On the Vulnerability of Low Entropy Masking Schemes. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2013. Berlin, Germany.

A Proofs of security property claimed in Theorem 2: resistance against jth-order ($j < d_{\mathcal{C}} = d_{\mathcal{D}}^{\perp}$) in the case of monovariate side-channel attacks

We denote in this appendix by Ψ the function that encodes x, i.e., $\Psi(x) = xG \in \mathbb{F}_2^n$. In ODSM, $\Psi(x)$ is manipulated masked by some $d \in \mathcal{D}$ (see Eq. (1)). The indicator of $\mathcal{D}$ is noted $f : \mathbb{F}_2^n \to \mathbb{F}_2$, in the sense that:

$$\forall d \in \mathbb{F}_2^n, \quad f(d) = 1 \iff d \in \mathcal{D} \ .$$

Said differently, $f(d) = 1 \iff \exists y \in \mathbb{F}_2^{n-k}$ s.t. $yH = d$.

For the rest of the security analysis, we resort to statistics. Thus, we use the following notations: capital letters (e.g., D) are random variables, small letters (e.g., d) are realizations, and calligraphic letters (e.g., $\mathcal{D}$) are representing the support of random variables.

Obviously, a monovariate first-order attack fails is the mask D is balanced. As motivated by monovariate high-order attacks coined by Moradi [18] and Carlet et al. [9] the attacker needs to create combinations between the bits of Z. Consequently, we model the attacker as a pseudo-Boolean function $\Phi : \mathbb{F}_2^n \to \mathbb{R}$ of a given numerical degree j in the bits of Z. For example, Φ can be the power j of the Hamming weight (as in zero-offset attacks). The leakage model can be, in general, any affine function of the bits of Z. This simply means that there is no "glitch" nor "cross-couping". This case is usual for software platforms.

So, when $j = 2$, the function Φ can model the product of two bits. This model captures the probing attacks, as with j probes, the attacker can build any polynomial of degree j in the sensible variables.

Proposition 3 (jth-order security condition on the masks coding). *Let $\Phi : \mathbb{F}_2^n \to \mathbb{R}$ a leakage function of numerical degree j, an arbitrary $\Psi : \mathbb{F}_2^k \to \mathbb{F}_2^n$ and a mask D uniformly distributed in a code $\mathcal{D}$, with f the indicator of $\mathcal{D} \subset \mathbb{F}_2^n$. Then the leakage $\Phi(\Psi(X) \oplus D)$ resists a monovariate attack if $\mathcal{D}$ is a code of dual distance $j + 1$.*

This Proposition 3 is the accurate rephrasing of Theorem 2. We aim now at proving them. In Proposition 3, the condition of jth-order security is: for all Φ of numerical degree smaller than or equal to j, $\mathbb{E}\left[\Phi(\Psi(X) \oplus D) | X = x\right]$ does not depend on $x \in \mathbb{F}_2^k$. This is rewritten as the condition:

$$\mathsf{Var}\left[\mathbb{E}\left[\Phi(\Psi(X) \oplus D) | X\right]\right] = 0 \ . \tag{14}$$

Indeed, in this case, any correlation attack fails: indeed, there is no linear dependency between the leakage $\Phi(\Psi(X) \oplus D)$ and the sensitive variable X.

Now, the expectation $\mathbb{E}\left[\Phi(\Psi(X)\oplus D)|X=x\right]$ is taken on the mask D random variable only, because $\Psi(X)$ depends only on X. So we have:

$$\begin{aligned}\mathbb{E}\left[\Phi(\Psi(X)\oplus D)|X=x\right] &= \sum_{d\in\mathcal{D}} \frac{1}{\mathsf{Card}[\mathcal{D}]}\Phi(\Psi(x)\oplus d)\\ &= 2^{-(n-k)}\sum_{d\in\mathbb{F}_2^n} f(d)\Phi(\Psi(x)\oplus d)\\ &= 2^{-(n-k)}\left(f\otimes\Phi\right)\left(\Psi(x)\right)\ .\end{aligned}$$

So, the countermeasure is jth-order secure if and only if $\left(f\otimes\Phi\right)\left(\Psi(x)\right)$ does not depend on x. Therefore, a sufficient condition for resistance against jth-order attacks is that $f\otimes\Phi(z)$ does not depend on $z\in\mathbb{F}_2^n$ (irrespective of function Ψ).

Let g a pseudo-Boolean function $\mathbb{F}_2^n\to\mathbb{R}$. We call $\hat{g}$ the Fourier transform of g, i.e., $\hat{g}(z)=\sum_a g(a)(-1)^{a\cdot z}$. We have: ($g$ is constant) $\iff \forall z\neq 0,\ \hat{g}(z)=0 \iff \hat{g}\propto\delta$, the Kronecker symbol.

Let us apply this result to $g=f\otimes\Phi$. The Fourier transform turns a *convolution product* into a *product*, i.e., $\widehat{f\otimes\Phi}(z)=\widehat{f}(z)\widehat{\Phi}(z)$. To prove that:

$$\widehat{f}\,\widehat{\Phi}=0\ , \tag{15}$$

let us introduce the following useful Lemma 1.

Lemma 1. *Let P be a pseudo-Boolean function $P:\mathbb{F}_2^n\to\mathbb{R}$ of numerical degree $d^\circ(P)$ [8,10]. Then, $\forall z\in\mathbb{F}_2^n$, $w_H(z)>d^\circ(P)\Longrightarrow\widehat{P}(z)=0$.*

Proof. Any pseudo-Boolean function can be written uniquely as a multilinear polynomial $P(y_1,\cdots,y_n)=\sum_{I\subseteq\mathcal{P}([\![1,n]\!])}a_I y^I$, where $\mathcal{P}([\![1,n]\!])$ is the set of all subsets of interval $[\![1,n]\!]$, a_I is a real coefficient, and y^I is an abbreviation for $\prod_{i\in I}y_i$. By definition, the numerical degree $\mathrm{d}^\circ(P)$ of P is the maximal degree of each monomial, *i.e.* $\mathrm{d}^\circ(P)=\max\{\mathsf{Card}[I]\text{ s.t. }a_I\neq 0\}$. By linearity of the Fourier transform, $\widehat{P}(z)=\sum_{I\subseteq\mathcal{P}([\![1,n]\!])}a_I\widehat{M_I}(z)$, where $M_I: y\in\mathbb{F}_2^n\mapsto y^I$. Let us prove that $\forall z\in\mathbb{F}_2^n$, $w_H(z)>\mathsf{Card}[I]\Longrightarrow\widehat{M_I}(z)=0$. Let z such that $w_H(z)>\mathsf{Card}[I]$. Thus z has at least one non-zero coordinate outside I. As all the coordinates in M_I are equivalent, we can assume (without loss of generality), that this coordinate is the last one. We note $y=(y',y_n)$, where $y'=(y_1,\cdots,y_{n-1})\in\mathbb{F}_2^{n-1}$ and $y_n\in\mathbb{F}_2$. Thus, $\widehat{M_I}(z)=\sum_{y'\in\mathbb{F}_2^{n-1}}\sum_{y_n\in\mathbb{F}_2}y^I(-1)^{y\cdot z}=\sum_{y'\in\mathbb{F}_2^{n-1}}(y',1)^I(-1)^{y'\cdot(z_1,\cdots,z_{n-1})}(1+(-1))=0$. As, by definition, any monomial M_I has numerical degree $\mathrm{d}^\circ(M_I)\leq\mathrm{d}^\circ(P)$, we also have $\widehat{P}(z)=\sum_{I\subseteq\mathcal{P}([\![1,n]\!])}a_I\times 0=0$. □

Based on Lemma 1, we give hereafter the proof of Proposition 3 / Theorem 2.

Proof. (Proof of Proposition 3) So, to prove that $\widehat{f}\,\widehat{\Phi}=0$, we start by applying Lemma 1. As Φ is of numerical degree j, $\widehat{\Phi}(z)=0$ for $w_H(z)>j$. So, the masking is jth-order secure if $\forall z\in\mathbb{F}_2^n$, $0<w_H(z)\leq j$, $\widehat{f}(z)=0$. By definition, this

means that f is jth-order correlation-immune (j-CI in brief). This is equivalent to saying the $\mathcal{D}$ is of dual distance $d^{\perp}_{\mathcal{D}} = j + 1$.

Irrespective of the way the sensitive variable $X \in \mathbb{F}_2^k$ is mapped (by function Ψ) onto $\mathbb{F}_2^n$, a sufficient condition for security against zero-offset attacks [28] of orders $1, 2, \cdots, j$ is that the mask D be distributed uniformly in $\mathcal{D}$, a code of dual distance $j+1$. Said differently, the lowest order j of a successful zero-offset attack is equal to the dual distance of $\mathcal{D}$.

As $\mathcal{D} = \mathcal{C}^{\perp}$, we have that $d^{\perp}_{\mathcal{D}} = d_{\mathcal{C}}$ (see Proposition 1). □

B Example of matrices for the ODSM on AES

The generator matrix G of $\mathcal{C}$ is written in systematic form in Eq. (16). This matrix is the direct result of the following Magma [25] command:

```
C := BestKnownLinearCode( FiniteField(2), 16, 8 );
```

The matrix H (see also Eq. (16)) is a basis of $\mathcal{C}^{\perp}$. As $G = [I_k \| M]$ is in systematic form, Proposition 2 can be readily applied to check whether the lines of G and of $H = [M^{\mathsf{T}} \| I_{n-k}]$, together, form a basis of $\mathbb{F}_2^{16}$. It happens that indeed, $I_8 \oplus MM^{\mathsf{T}}$ has rank 8, and so $\mathcal{C}^{\perp}$ is the supplementary of $\mathcal{C}$.

$$G = \begin{pmatrix} 1&0&0&0&0&0&0&0 & 1&0&0&1&1&1&1&0 \\ 0&1&0&0&0&0&0&0 & 0&1&0&0&1&1&1&1 \\ 0&0&1&0&0&0&0&0 & 1&1&0&0&1&1&0&0 \\ 0&0&0&1&0&0&0&0 & 0&1&1&0&0&1&1&0 \\ 0&0&0&0&1&0&0&0 & 0&0&1&1&0&0&1&1 \\ 0&0&0&0&0&1&0&0 & 1&1&1&1&0&0&1&0 \\ 0&0&0&0&0&0&1&0 & 0&1&1&1&1&0&0&1 \\ 0&0&0&0&0&0&0&1 & 1&1&0&1&0&1&1&1 \end{pmatrix}, \quad H = \begin{pmatrix} 1&0&1&0&0&1&0&1 & 1&0&0&0&0&0&0&0 \\ 0&1&1&1&0&1&1&1 & 0&1&0&0&0&0&0&0 \\ 0&0&0&1&1&1&1&0 & 0&0&1&0&0&0&0&0 \\ 1&0&0&0&1&1&1&1 & 0&0&0&1&0&0&0&0 \\ 1&1&1&0&0&0&1&0 & 0&0&0&0&1&0&0&0 \\ 1&1&1&1&0&0&0&1 & 0&0&0&0&0&1&0&0 \\ 1&1&0&1&1&1&0&1 & 0&0&0&0&0&0&1&0 \\ 0&1&0&0&1&0&1&1 & 0&0&0&0&0&0&0&1 \end{pmatrix}. \tag{16}$$

The matrices involved in $z = xG \oplus yH \mapsto x$ and $z \mapsto y$ (see Eq. (3) and (4)) are given in Eq. (17).

$$G^{\mathsf{T}}(GG^{\mathsf{T}})^{-1} = \begin{pmatrix} 0&0&1&1&1&0&0&1 \\ 0&0&1&0&0&1&0&1 \\ 1&1&1&1&1&0&1&0 \\ 1&0&1&0&1&1&0&0 \\ 1&0&1&1&1&1&1&0 \\ 0&1&0&1&1&1&1&1 \\ 0&0&1&0&1&1&1&1 \\ 1&1&0&0&0&1&1&0 \\ 0&1&0&1&1&0&1&0 \\ 1&1&0&0&0&1&0&1 \\ 0&1&1&0&0&0&1&0 \\ 0&0&1&1&0&0&0&1 \\ 1&1&0&0&1&0&0&1 \\ 1&0&0&0&1&1&0&0 \\ 1&0&0&1&0&1&1&1 \\ 0&1&1&1&0&0&1&0 \end{pmatrix}, H^{\mathsf{T}}(HH^{\mathsf{T}})^{-1} = \begin{pmatrix} 0&1&0&0&1&1&1&0 \\ 1&1&1&0&1&0&0&1 \\ 0&0&1&1&0&0&0&1 \\ 1&0&0&1&0&0&1&1 \\ 1&0&0&0&1&1&0&0 \\ 0&1&0&0&0&1&1&0 \\ 1&0&1&0&0&0&1&1 \\ 0&1&0&1&1&0&1&0 \\ 1&1&1&0&0&0&1&1 \\ 1&0&1&1&0&1&0&0 \\ 1&1&0&1&1&0&1&0 \\ 0&1&1&0&1&1&0&1 \\ 0&0&1&1&1&1&0&1 \\ 0&1&0&1&1&0&1&1 \\ 1&0&1&0&0&1&1&0 \\ 1&0&0&1&1&1&0&1 \end{pmatrix}. \tag{17}$$

Recall that the `xtime` function of AES [19, Sec. 4.2.1] is the multiplication by X in $\mathbb{F}_2^8$, seen as the finite field $\mathbb{F}_{2^8} \equiv \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$. It is

a linear function, generated from this $k \times k$ (i.e., 8×8) matrix L:

$$L = \begin{pmatrix} 0&0&0&1&1&0&1&1 \\ 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \end{pmatrix} . \tag{18}$$

The masked `xtime` function can be computed using Eq. (8). The generating matrix L' is $n \times n$ (i.e., 16×16):

$$L' = \begin{pmatrix} 1&1&0&0&1&0&1&1 & 1&1&0&1&0&0&1&0 \\ 0&0&1&0&1&1&1&1 & 1&1&1&0&1&1&0&0 \\ 0&0&1&1&0&1&0&1 & 0&1&0&0&0&0&1&1 \\ 1&1&1&1&1&1&1&1 & 0&1&1&1&0&0&1&0 \\ 1&1&0&1&0&0&0&1 & 0&1&0&1&0&0&1&1 \\ 1&1&1&0&0&1&0&1 & 1&1&0&0&1&0&1&0 \\ 0&1&1&1&0&0&1&1 & 0&0&1&1&0&0&1&0 \\ 0&1&0&1&0&0&0&0 & 1&1&1&1&1&1&1&0 \\ 1&1&1&0&1&1&1&0 & 0&0&1&0&0&1&0&1 \\ 0&1&0&1&0&1&0&0 & 1&0&0&1&1&0&1&1 \\ 1&0&1&0&0&1&1&0 & 1&1&1&1&1&0&0&1 \\ 0&1&0&1&0&0&1&1 & 1&0&0&1&0&1&1&1 \\ 0&1&0&0&0&0&0&0 & 0&1&0&0&0&1&1&1 \\ 1&0&0&0&1&1&1&1 & 1&1&1&1&0&1&0&1 \\ 1&0&1&0&0&0&1&0 & 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&1&0 & 0&1&1&1&0&0&1&0 \end{pmatrix} . \tag{19}$$

The Magma code to generate these matrices is listed in Fig. 1.

```
C := BKLC( GF(2), 16, 8 );
D := Dual(C);
Dimension( C meet D ); // 0 => Supplementary is checked

G := GeneratorMatrix(C); // LHS of Eqn. (16), [I|M]
H := HorizontalJoin(     // RHS of Eqn. (16), [M^T|I]
     Transpose(ColumnSubmatrix(G,9,8)),
     DiagonalMatrix( GF(2), [1,1,1,1,1,1,1,1] ));

J := Transpose(G)*(G*Transpose(G))^-1; // LHS of Eqn. (17)
K := Transpose(H)*(H*Transpose(H))^-1; // RHS of Eqn. (17)

// All those 5 checks are true:
G*Transpose(H) eq ZeroMatrix( GF(2), 8, 8 );
G*J eq DiagonalMatrix( GF(2), [ 1,1,1,1,1,1,1,1 ] );
G*K eq ZeroMatrix( GF(2), 8, 8 );
H*J eq ZeroMatrix( GF(2), 8, 8 );
H*K eq DiagonalMatrix( GF(2), [ 1,1,1,1,1,1,1,1 ] );

L := Matrix( GF(2), 8, 8, [
     0,0,0,1,1,0,1,1,
     1,0,0,0,0,0,0,0,
     0,1,0,0,0,0,0,0,
     0,0,1,0,0,0,0,0,
     0,0,0,1,0,0,0,0,
     0,0,0,0,1,0,0,0,
     0,0,0,0,0,1,0,0,
     0,0,0,0,0,0,1,0]); // Eqn. (18)
Lp := J*L*G + K*H;      // Eqn. (19)
```

Fig. 1. Magma code to compute generating matrices of supplementary dual codes.