A masking method based on orthonormal spaces, protecting several bytes against both SCA and FIA with a reduced cost

Claude Carlet *,† Abderrahman Daif ‡, Sylvain Guilley §,¶ Cédric Tavernier ‖

November 11, 2023

Abstract

In the attacker models of Side-Channel Attacks (SCA) and Fault Injection Attacks (FIA), the opponent has access to a noisy version of the internal behavior of the hardware. Since the end of the nineties, many works have shown that this type of attacks constitutes a serious threat to cryptosystems implemented in embedded devices. In the state-of-the-art, there exist several countermeasures to protect symmetric encryption (especially AES-128). Most of them protect only against one of these two attacks (either SCA or FIA). The main known countermeasure against SCA is masking; it makes the complexity of SCA growing exponentially with its order d. The most general version of masking is based on error correcting codes. It has the advantage of offering in principle a protection against both types of attacks (SCA and FIA), but all the functions implemented in the algorithm need to be masked accordingly, and this is not a simple task in general. We propose a particular version of such construction that has several advantages: it has a very low computation complexity, it offers a concrete protection against both SCA and FIA, and finally it allows flexibility: being not specifically dedicated to AES, it can be applied to any block cipher with any S-boxes. In the state-of-art, masking schemes all come with pros and cons concerning the different types of complexity (time, memory, amount of randomness). Our masking scheme concretely achieves the complexity of the best known scheme, for each complexity type.

Keywords: Masking countermeasure • Error correcting codes • Generalized Reed-Solomon codes • Orthonormal basis • Side-channel attack • Fault injection attack • Combined countermeasure • AES.

1 Introduction

When an algorithm is implemented on a hardware device (Chip card, TPM, FPGA, ...), the observable physical leakage (computing time, current consumption, electromagnetic radiation ...)

*University of Bergen, 5005, Norway.

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

‡BULL-SAS, Les Clayes-sous-Bois, France.

§Secure-IC S.A.S., Paris, France.

¶Telecom Paris, Institut Polytechnique de Paris, Palaiseau, France.

‖Hensoldt France, Plaisir, France.

can be exploited to mount so-called *side-channel attacks* (SCA). The most common countermeasures to combat such attacks are *masking* [GP99, CJRR99] and *shuffling* [RPD09]. *Shuffling* is a simple solution that involves randomizing a series of operations of the cipher so as to improve the SCA resistance, however, some advanced SCA techniques exist that break this countermeasure [CCD00, Mes00]. On the other hand, *masking* protects by mixing the sensitive data with some random value called the *mask*. The most generic known measure to protect against these attacks remains masking via homomorphic functions (see section 1.3 for a definition of this term). However, it is still a challenging matter to build such function which at the same time is not intensive in terms of computation, so that it can be implemented on low-resource electrical components, and also passes all constitutive operations of a symmetric encryption, in particular the substitution-Boxes (in brief S-boxes, e.g. `SubBytes` for AES).

Another type of attack, that threatens the electrical components, is called fault injection attack (FIA). It consists in disrupting the operation of encryption or decryption by the injection of malicious faults into a cryptographic device and the observation of the corresponding erroneous outputs [AK97, BDL01]. Despite the high cost of equipment used in this type of attacks, it remains the most effective for obtaining information about the sensitive data. However, attacks vary depending on the type of cryptography targeted (symmetric or asymmetric) [BBKN12].

Recently, side-channel and fault injection attacks have gained momentum in the fields of cloud computing. Indeed, it has been found several means to monitor timing side-channels [RMB15] and even the power side-channel of chips [LKO+21], from remote access. Regarding perturbation attacks, the Dynamic Voltage & Frequency Scaling (DVFS) feature has been leveraged to place the processor in a state such that it is hard for it to function normally, hence random faults occurring. Actual attacks have been reported and characterized, such as Plunder Volt [MOG+20]. In this respect, software implementations of cryptographic algorithms need to be protected by design against SCA and FIA.

Besides, for practical adoption in industrial products, the computational efficiency of the scheme is also important.

1.1 Related works

There exist several solutions to protect symmetric encryption from SCA by masking. The most conventional masking method is to decompose the sensitive data x into several parts (shares) $x_0, x_1, \ldots, x_d$ such that $x = \bigoplus_{i=0}^{d} x_i$, then operate on each of the parts separately without involving the sensitive data in the calculation process. This is called Boolean masking. Each $x \mapsto F(x)$ transformation that composes the encryption (or decryption) algorithm must be replaced by a function $(x_0, \ldots, x_d) \mapsto (y_0, \ldots, y_d)$ such that $F(\sum_{i=0}^{d} x_i) = \sum_{i=0}^{d} y_i$ (which is called *masking* by abuse of language), and such that the knowledge of d shares manipulated when calculating this function gives no information about x (which is known as d -th order probing security). This method remains efficient and simple when linear transformations (XOR, squaring in a field of characteristic 2) are applied to the data; however, it is still greedy in terms of calculations for nonlinear functions like `SubBytes` in AES. Since every function on a finite field is a polynomial, it suffices to know how to mask the addition (XOR) and multiplication. The difficulty is to build shares $c_0, \ldots, c_d$ such that $\bigoplus_{i=0}^{d} c_i = ab$. Ishai *et al.* [ISW03] implemented a solution which consists in securing the "NOT" and "AND" operations in a Boolean circuit. If we consider two sensitive bits $b = \bigoplus_{i=0}^{d} b_i$ and $b' = \bigoplus_{i=0}^{d} b'_i$, we can compute $\neg b = \neg b_0 \oplus \bigoplus_{i=1}^{d} b_i$ (" $\neg$ " denotes "NOT") and $bb' = \bigoplus_{i=0}^{d} \bigoplus_{j=0}^{d} b_i b'_j$. The calculation of bb' involves random values in order to secure it, this solution makes it possible to obtain a d^{th}-order security level [RP10], but the complexity in terms of computation and memory increases considerably according to the order. Prouff and Rivain [RP10] proposed a generalization of this algorithm, in particular in $\mathbb{F}_{2^8}$. This solution, dedicated to the AES, allows to reach an order as high as required. However, the quadratic complexity of the calculations remains quite greedy for the components endowed with little resources. The order is therefore limited to the supported capacity of the component.

Another method is the so-called "Polynomial Masking", introduced separately by Prouff-Roche [PR11] and Goubin-Martinelli [GM11], which combines Shamir's Secret Sharing Scheme (SSS) [Sha79] and secure multi-party computation techniques [BGW88]. The masking operation of a sensitive data $m \in \mathbb{F}_{2^8}$, consists in constructing a function of degree d, such that $f_m(x) = m \oplus \bigoplus_{i=1}^{d} a_i x^i$, where $(a_i)_{1 \leq i \leq d}$ are some random secret coefficients, then as in the previous scheme, m can be represented by d shares $(m_1, \ldots, m_d)$, with $m_i = (\alpha_i, f_m(\alpha_i))$ for $1 \leq i \leq d$ for some random inputs $(\alpha_i)_{1 \leq i \leq d}$. To get m (unmasked), we have to construct f_m *i.e.* calculate the coefficients $(a_i)_{1 \leq i \leq d}$) from $(\alpha_i, f_m(\alpha_i))_{1 \leq i \leq d}$ by polynomial interpolation, and finally calculate $m = f_m(0)$.

In [BCC+14] Bringer *et al.* are using linear codes with a complementary dual (also called LCD codes) to construct the Orthogonal Direct Sum Masking (ODSM). This allows the sensitive data to be masked with a random mask, chosen uniformly from a set of codewords. In this scheme, a sensitive data $x \in \mathbb{F}_2^k$ is associated with a codeword in a vector subspace $\mathcal{C} \subsetneq \mathbb{F}_{2^n}$. The codeword is then XORed with a random value from the dual of $\mathcal{C}$ and we obtain the masked value: $z := xG + rH$, with $G \in \mathbb{F}_2^{k \times n}$ being the generator matrix of $\mathcal{C}$, and $H \in \mathbb{F}_2^{(n-k) \times n}$ the parity-check matrix of $\mathcal{C}$, that is, the generator matrix of the dual of $\mathcal{C}$ (denoted by $\mathcal{D}$). Moreover, the vector spaces $\mathcal{C}$ and $\mathcal{D}$ are supplementary, *i.e.* $\mathcal{C} \oplus \mathcal{D} = \mathbb{F}_2^n$, this means that $\forall z \in \mathbb{F}_2^n$, $\exists!(x, r) \in \mathcal{C} \times \mathcal{D} \mid z = xG + rH$. To recover the sensitive data x from z, it is sufficient to calculate $zG^\top(GG^\top)^{-1}$. This scheme resists univariate attacks of degree $d_\mathcal{C} - 1$ ($d_\mathcal{C}$ denotes the minimum distance of $\mathcal{C}$) without increasing considerably the memory space used. Besides, this scheme allows to detect faults. We must distinguish two types of faults:

Definition 1 (Fault injection models)**.** *We consider two fault injection capabilities:*

1. ***Random faults****: the attacker is able to disrupt all the bits in the codeword, however without control. In this case, the fault can be modeled as the addition of a random $\epsilon \in \mathbb{F}_2^n$ uniformly distributed over $(\mathbb{F}_2^n)^*$.*

2. ***Low weight faults****: the attacker is limited in the amount of perturbation he can induce. In particular, his faults are restricted to low Hamming weight errors ϵ. Namely, there exists a nonzero integer d such that ϵ is uniformly distributed over set $\{\epsilon \in \mathbb{F}_2^n | 0 < w_H(\epsilon) \leq d\}$.*

We can characterize the fault detection rate of our scheme:

Lemma 1 (Detection rates)**.** *1. In the random fault model, the detection rate does not depend on the code properties. It is equal to $1 - (2^k - 1)/(2^n - 1)$.*

2. In the low weight fault model, all faults (i.e., 100% of them) are detected provided $d < d_\mathcal{C}$.

Proof. Let us start with the random fault model. As our scheme is computing on *codewords*, the only errors which are left undetected are those that turn a codeword into another codeword. Given that our codes are linear, this happens if and only if the error ϵ is a nonzero codeword. Hence, amongst the $2^n - 1$ possible errors (elements of $(\mathbb{F}_2^n)^*$), $2^k - 1$ cannot be detected. The detection rate is the complement to 1 of the rate of non-detected errors, the latter one being equal to $(2^k - 1)/(2^n - 1)$.

Now, regarding the low weight fault model, it is known that any error $\epsilon \neq 0$ of Hamming weight strictly inferior to the code minimum distance $d_\mathcal{C}$ is not a codeword. Therefore, all such errors are caught. Hence a detection rate of 1 if $d < d_\mathcal{C}$. □

The ODSM scheme has been initially designed for bit-level operations Subsequently, it has been improved to manipulate bytes, in [CDGT19].

Later on, it has been noticed that the orthogonality of the linear codes respectively generated by G and H was only a way to simplify the situation but was not a requirement, and getting rid of it could enable to derive codes with better parameters in the so-called DSM. The first paper in this respect is [CGM19], but operates on bits. The DSM scheme is extended to bytes in [WMCS20], where the detection and correction of faults is also sketched.

In [ABCV17], Azzi *et al.* presented a countermeasure against fault injection attacks entering in the framework of DSM but differently from [BCC$^+$14]. This method consists in encoding the sensitive data x using a systematic linear code. Let us consider $G = (I|A)$ the generator matrix of a linear code $\mathcal{C}$ in a systematic form (I denotes the identity matrix), $\texttt{Encode}(x) := xG = (x|xA)$ the encoding operation, and f a non linear transformation of AES (`SubBytes` for example). Before starting the encryption process, three tables $T0 : x \mapsto xA$, $T1 : x \mapsto f(x)$ and $T2 : xA \mapsto f(x)A$ must be pre-filled. Thus, using these tables, we can compute $(x|xA) \mapsto (f(x)|f(x)A)$, and thanks to the added redundancy $(f(x)A)$ we can detect the error injections according to the capacity of the chosen code. This method makes it possible to detect errors; in addition, it is possible to combine it with existing masking methods by applying a mask to the three tables (*i.e.* instead of using x, we can use $x + r$). The masked version with this method is a special case of the DSM family, in which the sensitive data x and the random mask r are encoded using the same code ($z = xG + rG = (x + r)G$). The advantage compared to the previous construction (ODSM) is that this scheme makes it easier to decode, since the masked word is already a code word. On the other hand, the disadvantage is that the mask remains identical throughout the encryption process because the tables $T0, T1, T2$ depend on it.

The present paper introduces a novel masking scheme, we name "Multivariate Direct Sum Masking" (in brief, MDSM). Our scheme is in the continuity of ODSM [BCC$^+$14] and DSM [CDGT19], with a simplification of computations. Namely, MDSM additionally brings several major specificities that we shall detail below; it improves then these previous schemes regarding different facets of the security; in particular, it takes into account the multi probing measurement and faults that can appear everywhere in the design of the cipher algorithm. MDSM completes also with a lower complexity the *cost amortization* property of [WMCS20], whereby several bytes of information can be masked with one single masking operation.

A recent study [WCG$^+$22] shows that the method in [WMCS20] is up to its security promises, in terms of actual side-channel analyses, and also optimizes its execution time, by simplifying the arithmetics. In our paper, we also optimize the masking scheme but at its structural level, by leveraging properties of its underlying codes. As are the schemes of [PR11, GM11], it is a d^{th}-order masking.

1.2 Contributions

In this paper, we solve two problems related to concomitant side-channel and fault injection attacks protection. Namely, we complement a masking scheme by a fault detection capability thanks to a super-encoding of the masked codewords in a one-phase countermeasure exploiting the fact that the dimension of the code being smaller than its length, the resulting redundancy allows error detection. The randomisation which brings masking against side-channel attacks and the redundant encoding which brings detection/correction capability against fault injection attacks are then *builtin* our countermeasure, which readily features at once both capabilities. This methodology allows for a verification of masked codewords integrity by a syndrome decoding algorithm, that can correct errors and erasures without compromising the objective of information protection by encoding with code-based random masking. Second, a specificity of our masking method is to exploit orthonormality, which allows to improve the computational efficiency. Indeed, computational speed shall be maximized despite the wealth of instantiated protections, and orthogonality also allows for algorithmic improvements in terms of memory size. The two requirements are captured as a need for three complementary space vectors, one for the information encoding, a second for the masking material, and a third for the fault detection capability. We detail error detection and correction algorithms which do not compromise the masking countermeasure. In addition to being in direct sum, we shall meet different constraints in respective terms of dimension, dual distance, and minimum distance. Furthermore, the existence of constructions of such triples of codes with the additional constraint of orthonormality is demonstrated.

1.3 Formalization

Our contribution consists in designing a d th-order software masking scheme of AES transformations, able to detect and correct errors that can be injected, and furthermore, minimize the costs in terms of memory and computing time as well.

When higher-order masking is involved, every sensitive variable s occurring during the computation is randomly split into $d+1$ shares $s_0, \ldots, s_d$ such that

$$s = s_0 \perp \ldots \perp s_d,$$

with $\perp$ a group law. Hence, we must prove that our scheme satisfies the following definition:

Definition 2. *(Masking d th-Order Soundness) [Mag12]: The masking Z of a sensitive variable s is sound at d th-order if:*

- *Z can be deterministically reconstructed knowing the d + 1 shares, while*
- *no information about Z can be extracted from strictly less than d + 1 shares.*

Let s and s' be two sensitive data, we denote the masked word of s by $\texttt{mask}(s)$, the challenge is to design two homomorphic functions `Add` and `SMult`, such that:

$$\begin{cases} \texttt{Add}(\texttt{mask}(s), \texttt{mask}(s')) = \texttt{mask}(s+s') \qquad \text{and} \\ \texttt{SMult}(\texttt{mask}(s), \texttt{mask}(s')) = \texttt{mask}(ss'). \end{cases}$$

With these two operations we can rebuild the masked version of any block cipher such as AES, and redefine each of its internal transformations (namely: `XOR, MixColumns, SubBytes`). In addition, to be able to detect and correct injected errors, the output space of the masking operation must be an error correcting code.

In [BCC+14] Bringer *et al.* use LCD codes, as recalled above. This allows the sensitive data to be masked with a random mask, chosen uniformly from a set of codewords. In addition, the unmasking does not need to store the chosen mask (except in the case where one wants to detect the errors). The approach that we will present in this paper is somewhat similar but more specific.

We have chosen to work on what we shall call a polynomial field, that is $\mathbb{F}_2[x]/p(x)$ where $p(x)$ is an irreducible polynomial. This allows using MDS codes (more information being then processed at the same time for the same correcting capacity), that is, codes whose minimum distance is optimal since it achieves the so-called Singleton bound with equality (the detecting and correcting capacity of such codes is then optimal). Indeed, it is necessary to work on $\mathbb{F}_{2^r}$ and not on $\mathbb{F}_2$, since MDS codes do not exist over $\mathbb{F}_2$, except ones with dimension or co-dimension at most 1, which do not present an interest.

Our method is also based on what we call orthonormal codes, that is, codes (over this field) having an orthonormal basis; this gives us more flexibility to build the homomorphic function and much more simplicity.

In our scheme, the same operation allows to mask and encode the sensitive information, as in [ABCV17], but this operation does not need to store the table of all possible inputs of the S-box as in the previous reference.

1.4 Outline

The rest of the paper is structured as follows:

- In section 2, we present a method to generate orthonormal MDS codes. The generator matrix of this code will be used as a parameter of our masking, in order to achieve the best performance in terms of complexity and security level.
- In section 3, we present in detail the masking method adopted, as well as all the operations that require masking (addition, multiplication, matrix product and exponentiation).
- In section 4, we provide the high-order security proof of our algorithms.
- In section 5, we provide attack results which show that MDSM offers no practical security reduction compared to its mother super-class [WMCS20].
- In section 6, we provide a comparison between our scheme and the state-of-the-art in terms of algorithmic complexity, performance and memory.
- In Section 7, we address the AES implementation.
- Section 8 is dedicated to the detection and correction of errors to protect against FIA.
- Eventually, an example of a triple of orthonormal codes construction is detailed in Appendix A.

2 Preliminaries

Let $K = (\mathbb{F}_2[x]/p(x), +, \cdot)$ be a polynomial field modulo an irreducible polynomial $p(x)$ of degree r (we can take in particular $r = 8$ and $p(x) = x^8 + x^4 + x^3 + x + 1$, the polynomial chosen for `SubBytes` and `MixColumns` transformations of the AES [Pub01]). Each equivalence class of this field (which is a representation of $\mathbb{F}_{2^r}$) is represented by a polynomial of degree at most $r - 1$ (for $r = 8$ the polynomial can be represented by two hexadecimal digits).

Given a positive integer m, we shall say that an $m \times m$ matrix E is self-orthonormal if its rows constitute an orthonormal family, that is, if E satisfies:

$$E \times E^\top = I_m \ , \tag{1}$$

where "$^\top$" denotes transposition and I_m is the $m \times m$ identity matrix. By abuse of language, we shall call self-orthonormal those codes admitting a self-orthonormal generator matrix. Let us take two sub-matrices of E (obtained by selecting two disjoint sets of rows in E), which we denote by $G \in K^{m_1 \times m}$ and $H \in K^{m_2 \times m}$, where $m_1 + m_2 \leq m$. The masking operation consists in calculating:

$$\texttt{mask}(\vec{s}) = \vec{s} \cdot G + \vec{r} \cdot H \ ,$$

for some sensitive data $\vec{s} \in K^{m_1}$ and a random mask $\vec{r} \in K^{m_2}$.

It has been shown in [PGS+17] that the order of the masking protection (in the probing security model) corresponds to the dual distance of $\mathcal{C}_H$ (the minimum distance of the dual of the code generated by H). For this reason, we must build E so that $\mathcal{C}_H^\perp$ is an MDS code and we know that this is equivalent to saying that $\mathcal{C}_H$ is MDS. Note that since E is orthonormal, $\mathcal{C}_H^\perp$ is generated by those rows of E that do not belong to H. Moreover, for better error detection and correction capacity, $\mathcal{C}_G + \mathcal{C}_H$ must also be an MDS code, since the error detection/correction will be ensured thanks to the code generated by the rows of E that are not chosen for G and H, and this code is MDS if and only if its dual $\mathcal{C}_G + \mathcal{C}_H$ is MDS.

2.1 MDS orthonormal codes, construction and properties

A self-dual code is a code equal to its dual code. Let us show how self-orthonormal matrices can be built from self-dual codes (which have been much studied and designed).

Proposition 1. *Let r, m be positive integers and let $\tilde{\mathcal{C}}$ be a linear code of length $2m$ and dimension m over K, with generator matrix (in systematic form) $G = [I_m|E] \in K^{m \times 2m}$. Then E is self-orthonormal if and only if the code $\tilde{\mathcal{C}}$ is self-dual.*

Proof. We have $G \times G^\top = I_m \times I_m + E \times E^\top$. We deduce that we have (1) if and only if $G \times G^\top = 0_m$ where 0_m is the $m \times m$ zero matrix. This latter property is equivalent to the fact that the code $\tilde{\mathcal{C}}$ is self-orthogonal, that is, satisfies $\tilde{\mathcal{C}} \subseteq \tilde{\mathcal{C}}^\perp$, and given its dimension, this is equivalent to the fact that it is self-dual, that is, satisfies $\tilde{\mathcal{C}} = \tilde{\mathcal{C}}^\perp$. □

Since any linear code admits a systematic generator matrix, we have then a construction of self-orthonormal matrices from self-dual codes. We know that self-dual $[2m, m]$ codes can be MDS, that is, have (optimal) minimum distance $m + 1$ (also equal to the dual distance), see [GG08].

Proposition 2. *Let r, m be positive integers and let $\tilde{\mathcal{C}}$ be a self-dual MDS code of length $2m$ over K (whose parameters are then $[2m, m, m+1]$), with generator matrix (in systematic form) $G = [I_m|E] \in K^{m \times 2m}$. Let us select m' rows in E, where $1 \leq m' \leq m$, obtaining an $m' \times m$ sub-matrix $E' \in K^{m' \times m}$ of E, and let us denote by $\mathcal{C}'$ the linear code over K generated by E'.*
Then, $\mathcal{C}'$ is an MDS orthonormal code of parameters $[m, m', m - m' + 1]$.
The dual $\mathcal{C}'^\perp$ of $\mathcal{C}'$ is also an MDS orthonormal code, with parameters $[m, m - m', m' + 1]$.

Proof. The codewords of $\mathcal{C}'$ being the linear combinations of m' rows of E, each of them can be obtained by:

- making a linear combination of m' rows of G,
- erasing the left half of the resulting codeword of $\tilde{\mathcal{C}}$.

Since this left half has Hamming weight at most m', the minimum distance of $\mathcal{C}'$ is at least $m+1-m'$ and the m' rows in E are linearly independent. Because of the Singleton bound, we have $d_{\mathcal{C}'} \leq m - m' + 1$ and we have then $d_{\mathcal{C}'} = m - m' + 1$, which means that $\mathcal{C}'$ is an MDS orthonormal code. The dual of $\mathcal{C}'$ is then an MDS code, since the dual of any MDS code is an MDS code, and it is also an orthonormal code, because E is orthonormal, and ${\mathcal{C}'}^{\perp}$ is obtained by the same process as $\mathcal{C}'$ by selecting the rows which have not been selected for building $\mathcal{C}'$. This completes the proof. $\square$

In [KL04] are given self-dual codes over $\mathbb{F}_{2^r}$, for $3 \leq r \leq 7$, of every even length n (equal here to $2m$) between 2 and 12 (which means that m can take any value between 1 and 6), which are:
- MDS (that is, with minimum distance $m+1$) for n between 2 and 8 (with an exception for $r = 3$ in the case of $n = 8$, where the code is only near MDS),
- near MDS (that is, with minimum distance m) for n equal to 10, 12 (with two exceptions: the codes are MDS for "$n = 10$, $r = 5$" and have minimum distance $m - 1$ only, for "$n = 12$, $r = 3$").

Remark 1. *Proposition 2 is more generally valid for a self-orthogonal (MDS)* $[n, m, d_{min} = n - m + 1]$ *code* $\tilde{\mathcal{C}}$ *with* $n \neq 2m$*. If* $\tilde{\mathcal{C}}$ *is obtained from a self-dual code by erasing rows from its generator matrix, then* E *in this new setting corresponds to* E' *in the former setting, and the parameters of the resulting code* $\mathcal{C}'$ *are* $[n - m, m', n - m - m' + 1]$.

2.2 Construction from GRS codes

Let $q = 2^r$. Let us choose n non-zero elements $v_1, \ldots, v_n$ in the finite field $K = \mathbb{F}_q$. Jointly, they are denoted $v = (v_1, \ldots, v_n)$. Besides, let $\alpha = (\alpha_1, \ldots, \alpha_n)$, where $\alpha_1, \ldots, \alpha_n \in K$ can be chosen arbitrarily. For $0 \leq k \leq n$, we define the *generalized Reed Solomon (GRS) code*:

$$GRS^q_{n,k}(\alpha, v) = \left\{(v_1 f(\alpha_1), \ldots, v_n f(\alpha_n)) \quad | \quad f \in K^k[X]\right\},$$

where $K^k[X]$ is the set of polynomials in $K[X]$ of degree less than k. It is well known that $GRS^q_{n,k}(\alpha, v)$ is an MDS code of length n, dimension k and minimum distance $n - k + 1$.
We recall also that the dual of $GRS^q_{n,k}(\alpha, v)$ is given by:

$$GRS^q_{n,k}(\alpha, v)^{\perp} = GRS^q_{n,k}(\alpha, u),$$

where $u = (u_1, \ldots, u_n)$ with $u_i^{-1} = v_i \prod_{j \neq i}(\alpha_i - \alpha_j)$.

Theorem 1. *([GG08]) There exist self-dual MDS codes* $\mathcal{C} = [2m, m, m+1]_q$ *over* K *for all* $m = 1, \ldots, 2^r - 1$.

We propose here to describe the construction of such code from [GG08]. Let $\tilde{\mathcal{C}} = GRS^q_{4m,m}(\alpha, v)$ be the generalized Reed-Solomon code of length $4m$, dimension m and minimum distance $3m + 1$ over K. We choose $v = \mathbf{1} = (1, \ldots, 1)$ and α is randomly chosen. Let $\tilde{G}$ be the generator matrix of $\tilde{\mathcal{C}}$. We denote by $M(\tilde{\mathcal{C}})$ the code generated by

$$M(\tilde{\mathcal{C}}) = \left\{\tilde{G}_i * \tilde{G}_j \mid i, j = 1, \ldots, m\right\}.$$

$M(\tilde{\mathcal{C}})$ is obviously a generalized Reed Solomon code $GRS^q_{4m,2m-1}(\alpha, \mathbf{1})$. We define then

$$P(\tilde{\mathcal{C}}) = M(\tilde{\mathcal{C}})^{\perp} = \langle \tilde{G}_i * \tilde{G}_j \mid i, j = 1, \ldots, m\rangle^{\perp}, \text{ with } \tilde{G}_i * \tilde{G}_j := (\tilde{G}_{i,1}\tilde{G}_{j,1}, \ldots, \tilde{G}_{i,4m}\tilde{G}_{j,4m}),$$

where $\tilde{G}_{i,l}$ denotes the entry in row i and column l of $\tilde{G}$. By construction $P(\tilde{\mathcal{C}}) = GRS^q_{4m,2m-1}(\alpha, u)$ where $u_i^{-1} = \prod_{j\neq i}(\alpha_i - \alpha_j)$. We can find a codeword $w^2 \in P(\tilde{\mathcal{C}})$ of weight $2m$ by using for example a Gaussian reduction algorithm. Since we have considered the field $K = \mathbb{F}_q$, w^2 can be written:

$$w^2 = (w_1^2, \ldots, w_{2m}^2), \text{ and } w = (w_1, \ldots, w_{2m}) \ .$$

Then for any codeword $\tilde{c}, \tilde{d} \in \tilde{\mathcal{C}}$, we have:

$$\langle w\tilde{c}, w\tilde{d}\rangle = \sum_{i=1}^{4m} w_i^2 \tilde{c}_i \tilde{d}_i = \langle v, \tilde{c} * \tilde{d}\rangle = 0 \ .$$

Thus the code $\mathcal{C}$ is obtained by puncturing $\tilde{\mathcal{C}}' = \{w * c \quad | \quad c \in \tilde{\mathcal{C}}\}$ at all positions where $w_i = 0$. We recall that $\tilde{\mathcal{C}} = GRS^q_{4m,m}(\alpha, \mathbf{1})$. Hence by construction, $\mathcal{C} = GRS^q_{2m,m}(\alpha', w)$ where the vector α' corresponds to the vector α restricted to the non-zero positions of w. Then it is a self-dual MDS code of length $2m$, dimension m and minimum distance $m + 1$.

2.3 Construction of an orthonormal basis and decoding

As explained in the above sections, a systematic generator matrix of a self-dual code

$$\mathcal{C} = GRS^q_{2m,m}(\alpha', w) \ ,$$

obtained by a Gaussian elimination leads to an orthonormal basis composed of the rows of E (see proposition 2). Then by construction the matrix E is also the generator matrix of a generalized Reed Solomon code:

$$GRS^q_{m,m}(\beta, w),$$

where the vector $\beta = (\alpha'_{m+1}, \ldots, \alpha'_{2m})$ comes from the vector $\alpha' = (\alpha'_1, \ldots, \alpha'_{2m})$.

Obviously, if we encode an information of dimension m over $\mathbb{F}_q$ with the generator matrix E of $GRS^q_{m,m}(\beta, w)$, we cannot correct any error since E is a full rank square matrix. We recall that Reed-Solomon codes are evaluation codes, then $\tilde{\mathcal{C}} = GRS^q_{4m,m}(\alpha, \mathbf{1})$ is generated by the monomials $1, X, \ldots, X^m$, then by cancelling the rows corresponding to $X^{k+1}, \ldots, X^m$, following the previous construction, we get an orthonormal matrix $E = GRS^q_{m,k}(\beta, w)$.

We provide in Alg. 1 the method to build an orthonormal basis. Note that this algorithm is executed only once in order to generate the initial masking parameters, that will be stored in the device memory.

Algorithm 1 `GenerateOrthoNormalMatrix`

Input: a parameter $m \in \mathbb{N}^*$
$\mathcal{C} \leftarrow GRS[4m, m, 3m+1]$
$G \leftarrow$ `systematicGeneratorMatrix`$(\mathcal{C})$
$\mathcal{C}' \leftarrow GRS[4m, 2m-1, 2m+1]$
$\mathcal{D} \leftarrow$ `dualCode`$(\mathcal{C}')$
$G' \leftarrow$ `systematicGeneratorMatrix`$(\mathcal{D})$
$\vec{v} \leftarrow G'_1$ — G'_1 is the first row of G'
$\vec{w} \in K^m$
for $1 \le i \le 4m$ **do**
    $w_i \leftarrow \sqrt{v_i}$ — *i.e.* $w_i^2 = v_i$
end for
$H \in K^{m \times 2m}$
for $1 \le i \le m$ **do**
    $H_{i,1} \leftarrow G_{i,1} * w_1$
    for $2 \le j \le 2m$ **do**
        $H_{i,j} \leftarrow G_{i,2m+j} * w_{2m+j}$
    end for
end for
$H' \leftarrow$ `SystematicForm`(H)
$E \in K^{m \times m}$
for $1 \le i \le m$ **do**
    for $1 \le j \le m$ **do**
        $E_{i,j} \leftarrow H'_{i,m+j}$
    end for
end for
return E

This algorithm meets the requirement of codes as specified in [WMCS20]. This paper only requires that the code generated by H be MDS, and the authors only provide one construction, namely from a Vandermonde matrix (See §6.1). Our algorithm is thus a special case of [WMCS20], which allows to compute faster.

3 The masking operations

Let $E \in K^{m \times m}$ be an orthonormal matrix of an MDS code, and let us denote by $G \in K^{m_1 \times m}$, $H \in K^{m_2 \times m}$ and $H' \in K^{m_3 \times m}$ three disjoint sub-matrices of E (*i.e.* $GH^\top = 0, GH'^\top = 0, HH'^\top = 0$) such that $m_1 + m_2 + m_3 = m$, which implies $GG^\top = I_{m_1}$, $HH^\top = I_{m_2}$, and $H'H'^\top = I_{m_3}$. and we denote by $\vec{s} = (s_1, s_2, \ldots, s_{m_1}) \in K^{m_1}$ a vector of sensitive data. The masking operation consists in generating a random mask $\vec{r} = (r_1, \ldots, r_{m_2}) \in K^{m_2}$ and calculating:

$$\texttt{mask}(\vec{s}) := \vec{s} \cdot G + \vec{r} \cdot H \ . \quad (2)$$

The third matrix H' will be used only for error detection and correction (see section 8).

Algorithm 2 `mask` complexity $\mathcal{O}(m(m_1 + m_2))$

Input: a sensitive data $\vec{s} \in K^{m_1}$
Output: $\vec{z} = \texttt{mask}(s) \in K^m$
$\vec{r} \xleftarrow{\$} K^{m_2}$ random mask
$\vec{z} \leftarrow \vec{0} \in K^m$
for $1 \leq i \leq m_1$ **do**
  for $1 \leq j \leq m$ **do**
    $z_j \leftarrow z_j + s_i G_{i,j}$
  end for
end for
for $1 \leq i \leq m_2$ **do**
  for $1 \leq j \leq m$ **do**
    $z_j \leftarrow z_j + r_i H_{i,j}$
  end for
end for
return $\vec{z}$

The algorithmic complexity of each function will be expressed in terms of the number of multiplications in K. In the following, we will denote $\vec{z} = \texttt{mask}(\vec{s}) = \vec{s} \cdot G + \vec{r} \cdot H$, and $\vec{z}\,' = \texttt{mask}(\vec{s}\,') = \vec{s}\,' \cdot G + \vec{r}\,' \cdot H$.
To extract the sensitive data $\vec{s}$ hidden in $\vec{z}$, we calculate:

$$\texttt{unmask}(\vec{z}) := \vec{z} \cdot G^\top \ . \tag{3}$$

Correctness.

$$\vec{z} \cdot G^\top \quad = \vec{s} \cdot \underbrace{(GG^\top)}_{=I_{m_1}} + \vec{r} \cdot \underbrace{(HG^\top)}_{=0} = \vec{s}\ .$$

Algorithm 3 `Unmask` $(\vec{z})$ complexity $\mathcal{O}(mm_1)$

Input: a masked value $\vec{z} = \texttt{mask}(\vec{s}) \in K^m$
Output: $\vec{s} \in K^{m_1}$
$\vec{s} \leftarrow 0 \in K^{m_1}$
for $1 \leq i \leq m_1$ **do**
  for $1 \leq j \leq m$ **do**
    $s_i \leftarrow s_i + z_j G_{i,j}$
  end for
end for
return $\vec{s}$

To proceed with the masking, we need to construct a homomorphic function for each of the operations that compose the symmetric cryptosystem, in particular the addition modulo 2 (XOR) and the multiplication over K.

3.1 Addition

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

$$\mathtt{mask}(\vec{s}+\vec{s}\,') = \mathtt{mask}(\vec{s}) + \mathtt{mask}(\vec{s}\,')\ . \tag{4}$$

Algorithm 4 `Add` $(\vec{z}, \vec{z}\,')$ complexity $\mathcal{O}(1)$

Input: $\vec{z} = \mathtt{mask}(\vec{s}), \vec{z}\,' = \mathtt{mask}(\vec{s}\,') \in K^m$
Output: $\vec{y} = \mathtt{mask}(\vec{s}+\vec{s}\,') \in K^m$
$\vec{y} \leftarrow 0 \in K^m$
for $1 \le i \le m$ **do**
    $y_i \leftarrow z_i + z'_i$
end for
return $\vec{y}$

3.2 Multiplication

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

3.2.1 Multiplication between public value and masked value

For this type of operations there is no need to mask the public coefficients. Thus, to mask an operation $\lambda \cdot \vec{s}$ for some public coefficient λ and sensitive data $\vec{s}$ we proceed thereby:

$$\begin{aligned} \lambda \cdot \mathtt{mask}(\vec{s}) &= (\lambda \cdot \vec{s}) \cdot G + (\lambda \cdot \vec{r}) \cdot H \\ &= \mathtt{mask}(\lambda \cdot \vec{s}). \end{aligned}$$

Algorithm 5 `Mult` $(\lambda, \vec{z})$ complexity $\mathcal{O}(m)$

Input: A public data $\lambda \in K$ and masked values $\vec{z} = \mathtt{mask}(\vec{s}) \in K^m$
Output: $\mathtt{mask}(\lambda \cdot \vec{s}) \in K^m$
$\vec{y} \leftarrow 0 \in K^m$
for $1 \le i \le m$ **do**
    $y_i \leftarrow \lambda z_i$
end for
return $\vec{y}$

For $m_1 > 1$ it is necessary to also define the matrix product, this type of transformation is essential to calculate `MixColumns` for example, with $m_1 \in \{4, 8, 12, 16\}$. Let us denote by $A \in K^{m_1 \times m_1}$ the public matrix, and denote $A' = G^\top A G + H^\top H \in K^{m \times m}$ we have:

$$\begin{aligned} \mathtt{mask}(\vec{s}) \cdot A' &= (\vec{s} \cdot G + \vec{r} \cdot H)(G^\top A G + H^\top H) \\ &= \vec{s} \cdot (G G^\top A G) + \vec{s} \cdot (G H^\top H) + \vec{r} \cdot (H G^\top A G) + \vec{r} \cdot (H H^\top H) \\ &= \vec{s} \cdot A G + \vec{r} \cdot H \\ &= \mathtt{mask}(\vec{s} \cdot A)\ . \end{aligned}$$

Algorithm 6 `MatrixProduct` $(\vec{z}, A')$ complexity $\mathcal{O}(m^2)$

Input: A masked values $\vec{z} = \texttt{mask}(\vec{s}) \in K^m$ and $A' = G^\top AG + H^\top H \in K^{m\times m}$
Output: $\texttt{mask}(\vec{s}\cdot A) \in K^m$. We assume that A' is a precomputed matrix.
$\vec{y} \leftarrow 0 \in K^m$
for $1 \leq i \leq m$ **do**
  for $1 \leq j \leq m$ **do**
    $y_i \leftarrow y_i + z_j A'_{j,i}$
  end for
end for
return $\vec{y}$

3.2.2 Multiplication between two masked values

Let us denote $\vec{z} = \texttt{mask}(s)$ and $\vec{z}\,' = \texttt{mask}(s')$. The multiplication algorithm consists in calculating $\texttt{mask}(\vec{s} * \vec{s}\,')$, where :

$$\vec{s} * \vec{s}\,' := (s_1 s'_1, s_2 s'_2, \ldots, s_{m_1} s'_{m_1}) \ ,$$

and '$*$' defines the point-wise multiplication between the elements of two vectors.

To do this multiplication, let us first build an algorithm that calculates the product by a single element of $\vec{s}$ in a specific position i, *i.e.* :

$$\begin{aligned} \texttt{OneProduct}(\texttt{mask}(\vec{s}), \texttt{mask}(\vec{s}\,'), i) &= \texttt{mask}(s_i \cdot \vec{s}\,') \\ &= \texttt{mask}(s_i s'_1, \ldots, s_i s'_{m_1}) \ . \end{aligned}$$

Let us denote by G_i, H_i the i^{th} row of G and H respectively, for $1 \leq i \leq m_1$ we have:

$$\begin{aligned} \vec{z}\cdot {G_i}^\top &= \textstyle\sum_{j=1}^m z_j G_{i,j} \\ &= s_i \ . \\ \Longrightarrow \quad \textstyle\sum_{j=1}^m z_j G_{i,j} \cdot \vec{z}\,' &= s_i \cdot \vec{z}\,' \\ &= \texttt{mask}(s_i \cdot \vec{s}\,') \end{aligned} \tag{5}$$

To securely compute $\sum_{j=1}^m z_j G_{i,j}$ without leakage, we choose to add a mask at each step of this sum, otherwise we disclose the sensitive information s_i. For $1 \leq k \leq m$ we have:

$$\begin{aligned} & \texttt{mask}(s_i \cdot \vec{s}\,')_k && = s_i z'_k \\ & && = \textstyle\sum_{j=1}^m s_j G_{i,j} z'_k \\ \Longrightarrow \quad & \texttt{mask}(s_i \cdot \vec{s}\,')_k + \underbrace{\sum_{j=1}^m r''_{(j\%m_2)+1} H_{(j\%m_2)+1,k}}_{\text{Mask refresh (i.e. } = \texttt{mask}(\vec{0})_k)} && = \textstyle\sum_{j=1}^m s_j G_{i,j} z'_k + r''_{(j\%m_2)+1} H_{(j\%m_2)+1,k} \\ \Longrightarrow \quad & \texttt{mask}(s_i \cdot \vec{s}\,')_k && = \textstyle\sum_{j=1}^m s_j G_{i,j} z'_k + r''_{(j\%m_2)+1} H_{(j\%m_2)+1,k} \ . \end{aligned} \tag{6}$$

This operation consists in unmasking in a secure way a single element of the 1st sensitive data and multiplying it by the masked value of the second one (using `Mult` algorithm). The sum $\sum_{j=1}^{m} z_j G_{i,j}$ discloses the sensitive element s_i, it is therefore important (to be high-order secure) to add a random mask $r''_j \cdot H_{(j\%m_2)+1}$ at each calculation step of this sum. Since H is matrix with m_2 rows, and since the refresh involves $1 \leq j \leq m$ rows of H, we need to use $(j\%m_2) + 1$ fresh random masks (where % denotes the modulo operation).

Algorithm 7 `OneProduct` $(\vec{z}, \vec{z}\,', i)$ — complexity $\mathcal{O}(m(2m+1))$

Input: Two masked values $\vec{z} = \texttt{mask}(\vec{s}), \vec{z}\,' = \texttt{mask}(\vec{s}\,') \in K^m$ and $i \in \{1, \dots, m_1\}$.
Output: $\texttt{mask}(s_i \cdot \vec{s}\,') \in K^m$
$\vec{y} = 0 \in K^m$
for $1 \leq j \leq m$ **do**
  $r''_j \xleftarrow{\$} K$
  $\lambda \leftarrow z_j G_{i,j}$
  for $1 \leq k \leq m$ **do**
    $y_k \leftarrow y_k + (\lambda z'_k + r''_j H_{(j\%m_2)+1,k})$
  end for
end for
return $\vec{y}$

Now, let us consider a three dimensions matrix $B \in K^{m_1 \times m_1 \times m_1}$ such that:

$$B_{i,j,k} = \begin{cases} 1 & \text{if } j = k = i \\ 0 & \text{otherwise} \end{cases}, \qquad \text{for } 1 \leq i, j, k \leq m_1$$

We have :

$$\vec{s} \cdot B_i \quad = \quad (0, \dots, 0, s_i, 0, \dots, 0) \; .$$

Thus, by using `MatrixProduct` algorithm with $B'_i = G^\top B_i G + H^\top H$ for $1 \leq i \leq m_1$, we obtain:

$$\begin{aligned} \texttt{MatrixProduct}(\vec{z}, B'_i) &= \texttt{mask}(\vec{s} \cdot B_i) \\ &= \texttt{mask}(0, \dots, s_i, \dots, 0) \; . \end{aligned}$$

By combining `OneProduct` and `MatrixProduct` we obtain, for $1 \leq i \leq m_1$:

$$\begin{aligned} \vec{v}_i &= \texttt{MatrixProduct}(\texttt{OneProduct}(\vec{z}, \vec{z}\,', i), B'_i) \\ &= \texttt{MatrixProduct}(\texttt{mask}(s_i s'_0, \dots, s_i s'_i, \dots, s_i s'_{m_1}), B'_i) \\ &= \texttt{mask}(0, \dots, 0, s_i s'_i, 0, \dots, 0) \; , \\ \text{which implies:} \quad \sum_{i=1}^{m_1} \vec{v}_i &= \texttt{mask}(s_0 s'_0, \dots, s_{m_1} s'_{m_1}) \\ &= \texttt{mask}(\vec{s} * \vec{s}') \; . \end{aligned}$$

Algorithm 8 `SMult` $(\vec{z}, \vec{z}\,')$ complexity $\mathcal{O}(m_1(3m^2+m))$

Input: Two masked values $\vec{z} = \texttt{mask}(\vec{s}), \vec{z}\,' = \texttt{mask}(\vec{s}\,') \in K^m$.
Output: $\texttt{mask}(\vec{s} * \vec{s}\,') \in K^m$
$\vec{y} = 0 \in K^m$
for $1 \leq i \leq m_1$ **do**
  $\vec{u} \leftarrow \texttt{OneProduct}(\vec{z}, \vec{z}\,', i)$ $= \texttt{mask}(s_i \cdot \vec{s}\,')$
  $\vec{v} \leftarrow \texttt{MatrixProduct}(\vec{u}, B'_i)$ $= \texttt{mask}(0, \ldots, 0, s_i s'_i, 0, \ldots, 0)$
  $\vec{y} \leftarrow \texttt{Add}(\vec{y}, \vec{v})$
end for
return $\vec{y}$

3.3 Exponentiation

The exponentiation allows to compute $\texttt{mask}(\vec{s}^{\,q})$, where $\vec{s}^{\,q} = (s_i^q)_{1 \leq i \leq m_1}$ from $\texttt{mask}(\vec{s})$ with q a power of 2. This allows to reduce considerably the computation complexity of the S-box. In fact, if we consider the AES case, the transformation `SubBytes` is composed only of 4 multiplications and 10 exponentiations (3 to calculate $\vec{s}^{\,-1} = \vec{s}^{\,254}$ and 7 to calculate the transformation (see Algorithm 2 in [PR11]).
Unfortunately, the exponentiation in codes based masking is not linear, in fact $\vec{z}^{\,q} = \vec{s}^{\,q} \cdot G^q + \vec{r} \cdot H^q$ where $(G^q)_{i,j} = (G_{i,j})^q$ and $(H^q)_{i,j} = (H_{i,j})^q$. However G^q and H^q keep the same properties as G and H (*i.e.* $G^q G^{q^\top} = I_{m_1}, H^q H^{q^\top} = I_{m_2}, G^q H^{q^\top} = 0$).

Proof. Let's denote $\vec{g}, \vec{g}\,' \in K^m$ two rows of G. We have:

$$\begin{aligned}
\textstyle\sum_{i=1}^m {g_i}^q {g'_i}^q &= \textstyle\sum_{i=1}^m (g_i g'_i)^q \\
&= \left(\textstyle\sum_{i=1}^m g_i g'_i\right)^q \qquad \text{since } q \text{ is a power of 2 and } K \text{ is a field of characteristic 2} \\
&= \begin{cases} 1 & \text{if } \vec{g} = \vec{g}\,' \\ 0 & \text{otherwise} \end{cases} \\
&= \begin{cases} 1 & \text{if } \vec{g}^{\,q} = \vec{g}^{\,\prime q} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

The same proof holds for H. □

Thus, it suffices to update the parameters to get back G and H. Let us denote : $Mq = G^{q^\top} G + H^{q^\top} H$, we have:

$$\begin{aligned}
\vec{z}^{\,q} \cdot Mq &= (\vec{s}^{\,q} \cdot G^q + \vec{r}^{\,q} \cdot H^q)(G^{q^\top} G + H^{q^\top} H) \\
&= \vec{s}^{\,q} \cdot \underbrace{G^q G^{q^\top}}_{=I_{m_1}} G + \vec{s}^{\,q} \cdot \underbrace{G^q H^{q^\top}}_{=0} H + \vec{r}^{\,q} \cdot \underbrace{H^q G^{q^\top}}_{=0} G + \vec{r}^{\,q} \cdot \underbrace{H^q H^{q^\top}}_{=I_{m_2}} H \\
&= \vec{s}^{\,q} \cdot G + \vec{r}^{\,q} \cdot H \\
&= \texttt{mask}(\vec{s}^{\,q}) \; .
\end{aligned}$$

Algorithm 9 `Exp` $(\vec{z}, q)$ complexity $\mathcal{O}(m^2)$

Input: A masked values $\vec{z} = \texttt{mask}(\vec{s}) \in K^m$ and $q \in \mathbb{N}$ a power of 2.
Output: $\texttt{mask}(\vec{s}^{\,q}) \in K^m$
$\vec{u} \leftarrow \vec{0} \in K^m$
$\vec{y} \leftarrow \vec{0} \in K^m$
for $1 \leq i \leq m$ **do**
    $u_i \leftarrow z_i^q$
end for
for $1 \leq i \leq m$ **do**
    for $1 \leq j \leq m$ **do**
        $y_i \leftarrow y_i + u_j * Mq_{i,j}$
    end for
end for
return $\vec{y}$

4 Security proof

The proof in Weijia Wang et al. [WMCS20] is checking for the strong non-interference (SNI) property as a whole. Our masking is a special case of [WMCS20], hence, their security proof applies verbatim to our masking. For the sake of being self-contained, let us recall its gist. This proof leverages the fact that encoding is carried out in a space vector of dual distance d. We recall that this means that every linear combination of $< d$ codewords is uniformly distributed (or 0 if the combination is trivial). The proof in [WMCS20] requires applying such property to various places in their Gadget 1. In our case, the proof applies similarly to Eqn. (5). In the rest of the paper our security model is SNI.

In the rest of this section, we provide a standalone proof of the d-th order security of our masking scheme.

The security shall be analyzed from two angles, namely:

1. the leakage of the *representation* (namely (2)), and
2. the leakage of the *operations*.

The representation is d-th order secure, as underlined in section 2. The proof regarding operations is more involved. Namely, there are two situations, depending on the operation:

- Combining two independent shares: it does not leak (by design);
- Combining two dependent shares requires special care. In this case, our solution consists in applying a constructive approach, as enunciated in Prop 3.

When combining two dependent shares, the following method allows to maintain the same security order:

Proposition 3 (Security order preservation by refresh)**.** *Security order is preserved when combining two shares according to this method:*

1. start by refreshing (at least) one of the shares, and

2. then proceed with the combination.

The proof is simply leveraging a security property of additive Boolean masking:

Proof. At step 1, the refresh allows to render independent the two shares. The result of step 1 can thus be securely combined with the other share(s). □

Notice that such refresh operation is commonplace in secure computation, and does not impede the computational efficiency from our MDSM scheme, which benefits the orthogonal structure of our masking parameters G and H.

All our linear operations leverage Prop. 3 each time there is a combination. For instance, this strategy is leveraged in Eqn. (6). During this operation, we assure that the operations are done in a secure way, considering our claim of dth-order side-channel SNI security.It consists in combining a transient unmasking a single element of the 1st sensitive data while simultaneously multiplying it by the masked value of the second one (using `Mult` algorithm). The sum $\sum_{j=1}^{m} z_j G_{i,j}$ would discloses the sensitive element s_i. It is therefore important (to be high-order secure) to add a random mask $r''_j \cdot H_{(j\%m_2)+1}$ at each calculation step of this sum.

The application of Prop. 3 might not be easy, e.g., in the case where a linear operation is applied on the masked representation of a variable. In such case, we propose a simple workaround. Assume that a constant matrix S is to be applied on a share z. Then, a d-th order secure strategy consists in splitting S into d random shares, nonetheless satisfying $S = \sum_{i=1}^{d} S_i$. Then, zS is evaluated as:

$$zS = (zS_1) + (zS_2) + \ldots + (zS_d). \tag{7}$$

This equation is to be used when computing a syndrome.

5 Attack results

5.1 Goal

Our MDSM masking scheme is a special case of that of Wang et al. [WMCS20]. Thus, we aim to check that MDSM is performing as good as that of Wang et al., in that the specific choice for the codes (orthogonality) does not undermine its security with respect to side-channel attacks. In particular, the security order is the same at word level[1], and we checked it is the same as well at bit level (after sub-field expansion).

Notice that Wang et al.'s masking scheme uses random generator matrices for codes C and D. Indeed, Wang et al.'s scheme is a generalization of Boolean Masking, Inner Product Masking, Direct Sum Masking, and Shamir Secret Sharing Masking. Our scheme is more constrained, since codes are built from a specific construction. Once more, we aim at proving that such construction does not lead to a particular leakage.

5.2 Methodology

In this respect, we simulated traces with identical parameters for Wang et al. and our masking, and some options, namely:

[1] Recall that we use "word" for symbols in $\mathbb{F}_{2^\ell}$, such as nibble (resp. bytes) when $r = 4$ (resp. $r = 8$), as opposition to bits.

Table 1: Parameters of the studied codes

Masking order	Wang et al.	Our MDSM scheme
1	$G = \begin{pmatrix} \texttt{0x4} & \texttt{0x2} & \texttt{0xB} \end{pmatrix}$ $H = \begin{pmatrix} \texttt{0x9} & \texttt{0x5} & \texttt{0x8} \end{pmatrix}$	$G = \begin{pmatrix} \texttt{0x6} & \texttt{0x4} & \texttt{0x3} \end{pmatrix}$ $H = \begin{pmatrix} \texttt{0x4} & \texttt{0xB} & \texttt{0xE} \end{pmatrix}$
2	$G = \begin{pmatrix} \texttt{0xC} & \texttt{0x5} & \texttt{0x2} & \texttt{0x4} \end{pmatrix}$ $H = \begin{pmatrix} \texttt{0x7} & \texttt{0xD} & \texttt{0x1} & \texttt{0xD} \\ \texttt{0x9} & \texttt{0x9} & \texttt{0xF} & \texttt{0x2} \end{pmatrix}$	$G = \begin{pmatrix} \texttt{0xA} & \texttt{0x4} & \texttt{0xD} & \texttt{0x2} \end{pmatrix}$ $H = \begin{pmatrix} \texttt{0x2} & \texttt{0xD} & \texttt{0x4} & \texttt{0xA} \\ \texttt{0xA} & \texttt{0xB} & \texttt{0x3} & \texttt{0x3} \end{pmatrix}$

- Simulation of one nibble ($r = 4$), shared with General Coding Masking and our (MDSM) orthogonal masking;
- Same fault detection capability of one code word;
- One mask (first order masking) and two masks (second order masking).

The generator matrices G and H for the C and D codes are given in Tab. 1, where the elements are constants in $\mathbb{F}_{16}$, represented as $\mathbb{F}_2[\alpha]/\langle\alpha^4 + \alpha + 1\rangle$. For example `0xC` in hexadecimal notation, or $(1100)_2$ in binary notation, represents the field element $\alpha^3 + \alpha^2$. The codes of generator matrix H have dual distance 2 (resp. 3) at word-level for $n = 3$ (resp. 4). At bit-level, the dual distances become 3 (resp. 4) for $n = 3$ (resp. 4). We even sought for codes which achieve similar in terms of side-channel protection. It is known [CGC+21] that the minimum distance is a parameter governing the security, but also the number of nonzero codewords of smallest nonzero weight (sometimes also referred to as the kissing number). Let us also introduce the weight distribution $(A_i)_{0\leq i\leq n}$ of a code. It is defined as:

$$\{A_0, A_1, \ldots, A_n\},$$

where $A_i = |\{c, w_H(c) = i\}|$, for $0 \leq i \leq n$. We therefore impose as well that the weight distribution of the code of Wang et al. and ours be the same, which is even more demanding than to have the same minimum weight and the same kissing number. Namely, the weight distributions we considered are equal to:

- $\{1, 0, 0, 17, 38, 44, 52, 54, 33, 12, 4, 1, 0\}$ for $n = 3$ and
- $\{1, 0, 0, 0, 4, 20, 36, 48, 45, 40, 36, 16, 6, 4, 0, 0, 0\}$ for $n = 4$.

The codes have been obtained by a construction implemented in SAGE (see the algorithm 1 and the implementation in Appendix A).

Traces are generated using the Hamming weight leakage model, each share leaking independently from the others.

For the sake of being unambiguous with respect to the attack, we opted for the most efficient attack, namely that based on maximum likelihood (see for instance Theorem 1 of [BGHR14]). It is innately a multivariate attack, which therefore leverages the n shares independently. We show in Fig. 1 and 2 the attack outcome for two noise variances $\sigma^2 \in \{1, 2\}$.

In those curves, the signal-to-noise ratio (SNR) is equal to σ^{-2}. Indeed, the information is the variance of $\mathcal{B}(r, \frac{1}{2})$, which is equal to $r/4 = 1$ for $r = 4$. Besides, the variance of the noise (for each share) is set to σ^2.

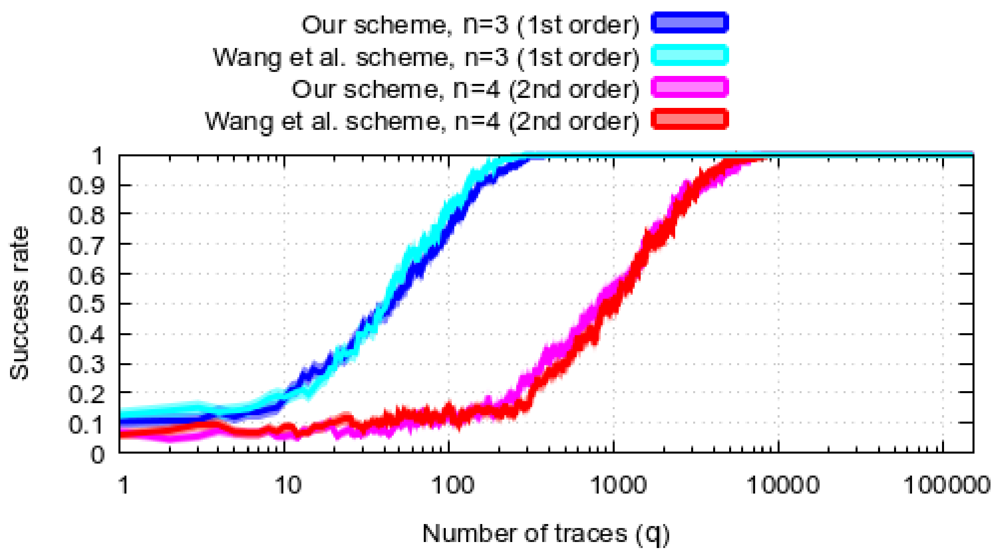

Figure 1: Attack result in terms of probability of success P_S, as function of the number of traces q, for an SNR= 1

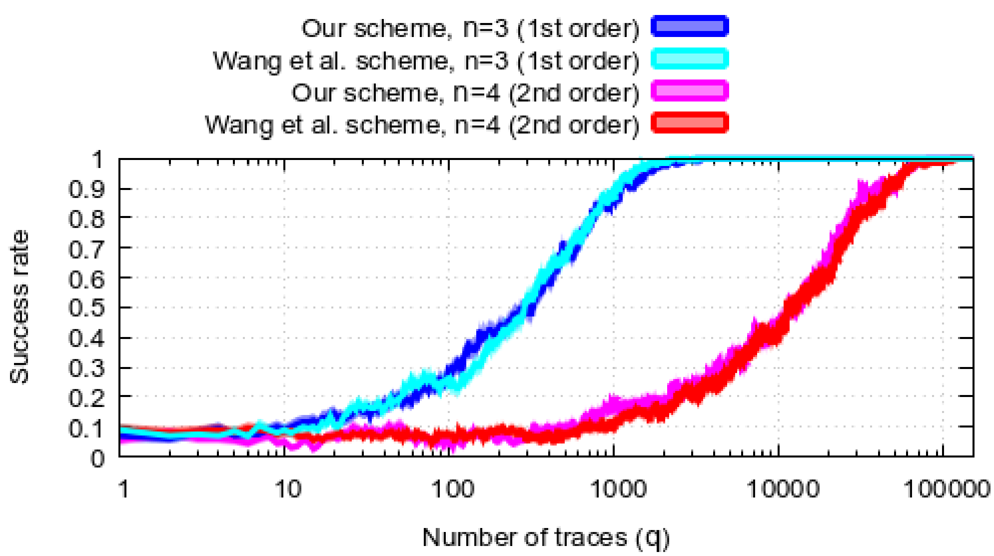

Figure 2: Attack result in terms of probability of success P_S, as function of the number of traces q, for an SNR= 1/2

In those plots, the error bars on the probability of success (denoted as P_S, and also known as the "success rate") are computed as [MRGD12]. The methodology leverages the fact the success rate is estimated as an empirical count of the number of success across different independent attacks. In our plots, 200 attacks are carried out, hence errors in interval $\left[-\sqrt{P_S(1-P_S)/200}, +\sqrt{P_S(1-P_S)/200}\right]$ around P_S.The fact that in Fig. 1 and 2 the success rate curves for [WMCS20] and MDSM schemes **do** overlap show that the resistance to attack is exactly the same, without regression arising from MDSM being a subclass of [WMCS20] masking scheme.

5.3 Conclusion of the section

The plots show that our masking scheme is as resistant as that of Wang et al. In this respect, the way we select the mask codes does not weaken the masking order. As we shall see, while maintaining the same security level, it also improves the computation speed.

6 Algorithmic complexity

Table 2 summarizes the algorithmic complexity of each of the algorithms we have presented. This complexity is calculated with respect to the size of the mask, the number of operations (addition, multiplication, exponentiation) performed in K, and the required number of random symbols.

Table 2: Complexity of each function from our masking scheme (encoding of m_1 sensitive variables)

	XORs	Multiplications	Exponentiation	Random
`mask`	$m(m_1+m_2)$	$m(m_1+m_2)$	0	m_2
`Unmask`	mm_1	mm_1	0	0
`Add`	m	0	0	0
`Mult`	0	m	0	0
`SMult`	$m_1(3m^2+m)$	$m_1(3m^2+m)$	0	m_1m
`Exp`	m^2	m^2	m	0

The algorithmic complexity of the masking depends on the complexity of the multiplication algorithm (`SMult`). This complexity is expressed as a function of the number of shares m, the masking order m_2 and the dimension of the sensitive data m_1. In Table 3 we present a comparison between our method and other state-of-the-art schemes in terms of the number of multiplications in K.

Remark 2. *We used as a quantum of complexity the number of multiplications in K, because, in hardware implementations, one typically reuses the same field multiplier irrespective one operand is statically known or not. Besides, in software, specializing multiplications for a given constant incurs code expansion. Notice that this estimation of complexity is not the one adopted in [WMCS20]: this paper only counts the multiplications between two statically unknown operands.*

Obviously, when the field is of small size, or when the number of constant multipliers is small, it can be beneficial to instanciate several specialized hardware multipliers. It is a matter of time vs area tradeoff. In this article, we adopt a univocal convention to quantify the complexity, by contemplating the case where all multiplications are executed by a single instance of a generic multiplier.

It can be seen that our algorithm has the lowest complexity compared to all other masking schemes.

Table 3: Secure multiplication complexity in time (number of multiplications) for single byte masking, comparison with state-of-the-art

	m_2	Memory complexity m	multiplication complexity	Fault C.
This paper	1	$m_2+1=2$	$3(m_2+1)^2+m_2+1=14$	
	2	$m_2+1=3$	$3(m_2+1)^2+m_1+1=30$	Yes
	3	$m_2+1=4$	$3(m_2+1)^2+m_2+1=52$	
	4	$m_2+1=5$	$3(m_2+1)^2+m_2+1=80$	
[PR11]	1	$2m_2+1=3$	$(2m_2+1)^3+(2m_2+1)=21$	
	2	$2m_2+1=5$	$(2m_2+1)^3+(2m_2+1)=80$	Yes
	3	$2m_2+1=7$	$(2m_2+1)^3+(2m_2+1)=203$	
	4	$2m_2+1=9$	$(2m_2+1)^3+(2m_2+1)=414$	
[BFG15]	1	$2m_2+1=3$	$3(2m_2+1)^2-(2m_2+1)=24$	
	2	$2m_2+1=5$	$3(2m_2+1)^2-(2m_2+1)=70$	No
	3	$2m_2+1=7$	$3(2m_2+1)^2-(2m_2+1)=140$	
	4	$2m_2+1=9$	$3(2m_2+1)^2-(2m_2+1)=234$	
[WMCS20]	1	$m_2+1=2$	$2(m_2+1)^3+(m_2+1)=20$	
	2	$m_2+1=3$	$2(m_2+1)^3+(m_2+1)=63$	Yes
	3	$m_2+1=4$	$2(m_2+1)^3+(m_2+1)=144$	
	4	$m_2+1=5$	$2(m_2+1)^3+(m_2+1)=275$	
[CGP+12]	2	$m_2(m_2+1)/2=3$	$(m_2+1)^2=9$	
	3	$m_2(m_2+1)/2=6$	$(m_2+1)^2=16$	No
	4	$m_2(m_2+1)/2=10$	$(m_2+1)^2=25$	

The nice property that our scheme has in common with [WMCS20] is that it is amenable to concomitant masking of m_1 sensitive variables. This allows to save randomness for refresh and to factor them across all m_1 sensitive variables. This results in optimized complexity in terms of number of field multiplication. Notice that, for a given $m_1 > 1$, our complexity is quadratic in the number of shares m. This is faster than [WMCS20] which is cubic in m. Namely, when no error detection is supported, *i.e.*, $m = m_1 + m_2$, one has:

- a complexity in $3m_1m^2 = 3m_1(m_1 + m_2)^2$ for our scheme, and
- a complexity in $2m^3 = 2(m_1 + m_2)^3$ for the scheme [WMCS20].

The fact that our scheme (and that in [WMCS20]) allows to masking multiple ($m_1 > 1$) bytes altogether has another byproduct: the complexity in memory usage does scale slower than proportionally with m_1. Indeed, the transmission rate of the code is equal to m_1/m (whereas it is $1/m$ when each sensitive variable is processed on its own).

7 AES implementation

7.1 Rationale

The AES block cipher can be computed end-to-end by applying transformations to each bytes in the state. Now, a challenge happens when some bytes of the state are masked together, *i.e.*, when $m_1 > 1$. As a matter of fact, some operations (e.g., MixColumns) combine several bytes together; there are thus two situations to consider:

- the bytes to combine pertain to different sharings, e.g., $z = xG + yH$ and $z' = x'G + y'H$; or
- the bytes to combine pertain to the very same sharings, e.g., they are two bytes within x, which is protected as $z = xG + yH$, where y is shared mask.

The first case allows to compute on masked shares as in any masking scheme. The second case requires to adapt the computation – actually, the processing of shares within a sharing can lead to demasking. We therefore detail in very case below.

Given that AES is also "four-byte" oriented, it is reasonable to envision $m_1 = 4$ or $m_1 = 16$ when taking advantage of privacy amplification. Indeed, such structure is amenable to operations such as MixColumns, which are column-oriented. Notice that in usual block ciphers, for reasons of speed, nonlinear operations are implemented as S-Boxes, which process bytes individually. Consequently, let us show how to evaluate a linear function L operating on 4-bytes $x = (x_1, x_2, x_3, x_4)$ that are masked together.

The linear operation $L : \mathbb{F}_{256}^4 \to \mathbb{F}_{256}^4$ can be represented as a row-matrix multiplication, namely

$$L(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4)M,$$

where M in a 4×4 square matrix representing the application of MixColumns on a single column. Let T the $n \times n$ square matrix defined as

$$T = [G; H]^{-1} \times [MG; H],$$

where $[A; B]$ denoted the vertical concatenation of rectangle matrices A and B sharing the same number of columns. This matrix shall be precomputed. It is straightforward to verify that $z' = zT = L(x_1, x_2, x_3, x_4)G + (y_1, ...)H$ is representing the masked version of MixColumns applied on (x_1, x_2, x_3, x_4). Let us remark that the row-matrix multiplication $z' = zT$ can advantageously be skipped, in a view to procrastinate. Effectively, if the computation is carried on z instead of z', the remainder of the computation remains correct (both from functionality and security standpoints), provided it is assumed that the masking matrices are no longer the pair (G, H) but the pair (MG, H). Of course, this applies only if MG remains orthonormal, which is the case for instance when dealing with ShiftRows operation.

7.2 Performance results

In order to assess the performance of MDSM, an implementation has been developed in C language. The same software code can handle different matrices; we chose to compare the computation speed for different masking configurations, albeit with the same security order $d = m_2 - 1 = 3$.

The table 4 reports a comparison in terms of calculation time and masked word size (m_1 values) during the calculation of the SubBytes transformation. The time is measured for the repetition of 500 transformations of "SubBytes" type. It can be seen that computation time decreased when m_1 increases. Those results are consistent with the complexity shown in Tab. 2 for `SMult`, namely, the computation time scales linearly with `SMult` complexity multiplied by the number of codewords ($m_0 = \frac{16}{m_1}$). Further speed-up can be gained when m_1 grows by leveraging *intrinsics* (computation using several bytes in parallel in a single machine word). To mask the 16 bytes using a masking of dimension $m_1 = 1$, each byte must be masked separately and we obtain a total of $m_0 = 16$ masked blocks of 23 bytes, which make a total of 368 bytes. In the other hand, only 23 is needed with $m_1 = 16$.

Table 4: Comparison of performance in terms of computation time and data size using different parameters and a same security order during the SubBytes calculation.

m	23				
m_1	1	4	8	16	
m_2	4				
m_3	18	15	11	3	$= m - (m_1 + m_2)$
m_0	16	4	2	1	$= 16/m_1$
Size	368	92	46	23	$= m_0 m$
Random	1472	1472	1472	1472	$= 64m$
Time (s)	16.88	10.43	9.72	9.02	

This use-case illustrates that for a given side-channel security order (m_2 is fixed), it is beneficial in terms of computational speed to process as many bytes of information as possible at the same time (computation time decreases when m_1 increases). Of course, this improvement comes at the expense of a lower fault detection capability, as m_3 decreases when m_1 increases (namely: $m_1 + m_3$ is constant $= m - m_2$).

8 Detection and correction of errors

Fault attacks are very efficient in general [JT12]. Some fault attacks, such as Statistical Ineffective Fault Attacks (SIFA [DEG+18], inheriting from the seminal work of [YJ00]) can be applied despite masking against side-channel analysis and fault detection mechanisms are in place.

We considered two representative fault models, namely one where the attacker has no control over the fault (random model), and one where the attacker can inject targeted low weight faults. We recall that, in front of uniformly random faults, the detection capability is only characterized by the code co-dimension m_3. The detection is more subtile in front of low weight faults, as the minimum distance of code is involved. We develope this case in more details for this reason.

We assume that the attacker has the ability to inject a certain number of simultaneous faults which is less than the correction capacity of the considered code. We detail bellow the features of our code. We consider also that all codewords present in the implementation are corrected/checked. If not, we face an open problem: the impact of the error propagation in the cipher algorithm design and this is out of scope of this paper.

By construction, according to Subsection 2.3, each masked element belongs to the code $E = GRS^q_{m,k}(\beta, w)$. Intentional or accidental errors can disturb the symmetric cipher implementation. If an error appears during the first rounds of the considered cipher, then its propagation shall affect dramatically the rest of calculation, making the final result wrong and uncorrectable due to the excessive number of errors, or can give information (for fault attacks) that may compromise the key. Such scenarios appear for example in case of radiation or in case of intentional fault attacks. We are also aware that such channel perturbation can lead to the presence of erasures, which means that information simply disappears. As we must consider the problem of decoding generalized Reed-Solomon codes, erasures simply mean that certain positions of the vector w are zero. Hence, a decoding algorithm that works for Generalized Reed Solomon codes can correct erasures. Of course it is essential that our counter-measure against FIA does not weaken the counter-measure against SCA, thus we propose to show in this section that syndrome decoding cannot leak information.

Namely, we offer the possibility of either detecting or even correcting errors and erasures anywhere in the calculation process where codewords are available. Generalized Reed-Solomon codes have the good property to support adversary channels which means that asymptotically almost all errors are correctable beyond the decoding capacity. In general, decoding errors leads to unmask the sensitive information, which is of course not desired between the first and last round of the algorithm that we must protect. For example, Sudan [GS99] and Berlekamp-Welch [RR86] algorithms return directly the sensitive information, while syndrome decoding does not.

Decoding generalized Reed-Solomon codes is well known, but we are particularly interested in syndrome decoding which does not reveal any sensitive information. The algorithm [Sha07, McE77, KB10] that uses the Euclidean algorithm is a syndrome decoding algorithm. It consists in building the polynomials that correspond to the error evaluator and error locator as explained in Theorem 4.3 of [Sha07]. Hence, this algorithm returns the vector corresponding to the error, that allows to return the corrected codeword belonging to the generalized Reed-Solomon code.

Irrespective of the decoding algorithm, it is noteworthy that never the sensitive information is exposed during the process of decoding because the first step consists in cancelling the codeword coming from the encoded information in order to construct the error as shown below.

In section 3 we have seen how to construct our masking parameters G, H, and H' from E. We have:

$$\begin{aligned} GH'^{\top} = 0 \quad \text{and} \quad HH'^{\top} &= 0 \\ \Longrightarrow \qquad (\vec{s}\cdot G + \vec{r}\cdot H)H'^{\top} &= 0 \qquad \forall \vec{s} \in K^{m_1}, \quad \forall \vec{r} \in K^{m_2} \ . \end{aligned}$$

Thus, by a simple syndrome calculation, if we suppose $\vec{z}$ was modified by a fault injection attack or a radiation, then we get $\vec{z}\,' = \vec{z} + \vec{e}$, and we have:

$$\vec{\epsilon} = \vec{z}\,' \cdot H'^{\top} = \vec{z} \cdot H'^{\top} + \vec{e} \cdot H'^{\top} = \vec{e} \cdot H'^{\top} \ .$$

Obviously the syndrome calculation does not bring any information since by definition a codeword corresponds to information that has been masked and we have assumed that the potential attacker has not more than d probes, thus no linear transformation can provide any information on the sensitive information.

We note however that determining the efficiency of this method when faults take place in the decoding algorithm itself remains an open problem. But the method is efficient when the fault injections are directed on the masked design of the ciphered algorithm. Then each variable being encoded by our generalized Reed-Solomon code, we may potentially check all variables (this has of course a non negligible cost). The attacker may inject faults on the matrices G and H to disturb the multiplication; then either the number of constructed errors is too large and the algorithm cannot correct it, but it simply detects and alerts (key zeroization...), or the number of errors is reasonable and the error correction algorithm can correct the disturbed multiplication.

Remark 3. *A legitimate question comes from the possibility of other kinds of side channel attacks during the syndrome decoding. According to the refresh procedure that we apply in proposition 3, power analysis is very difficult.*

The only way for an attacker seems to produce a fault that transforms a codeword in another codeword. We evaluate now the probability to succeed in this operation.

Theorem 2. *[PGS+17] Let C a $[n, k, d]$ MDS code over $\mathbb{F}_q$. Let $w \in [0, n]$. Then the number of weight w codewords belonging to C is:*

$$A_w = \binom{n}{w} \sum_{j=0}^{w-d} (-1)^j \binom{w}{j} (q^{w-d+1-j} - 1) = \binom{n}{w} (q-1) \sum_{j=0}^{w-d} (-1)^j \binom{w-1}{j} q^{w-d-j} \ .$$

Let I be the injected fault on a codeword c such that $I \oplus c$ still belongs to the codeword. If $a = c \oplus I$ is a codeword, it means by definition of C that $weight(I) \geq d$. We have $\binom{n}{d}(q-1)^d$ faults of weight d, thus the probability to pick up an undetected fault is not more than $\frac{A_d}{\binom{n}{d}(q-1)^d} = \frac{1}{(q-1)^{d-1}}$. We can assume that the attacker picks up randomly a fault of weight $\in [d, n]$, then the probability of success is:

$$\sum_{i=d}^{n} \frac{A_i}{(q-1)^d \binom{n}{i}}.$$

A numerical evaluation of this quantity allows us to conclude that our scheme is almost fault immune against fault injection attacks.

Remark 4. *Let $\mathcal{C}_{F'}$ be the space vector generated by F'. We note that syndromes ϵ belong to $\mathcal{C}_{F'}$. However $\mathcal{C}_{F'}$ meets with $\mathcal{C}_G$ (the space vector containing the sensitive information) only in zero. Thus the syndrome calculation does not reveal any information on the sensitive data. Actually $\epsilon \notin \mathcal{C}_G \oplus \mathcal{C}_H$.*

Following our notation, we have a code dimension $k = m_1 + m_2$ and a code length m. Thus, according to [KB10], decoding generalized Reed-Solomon codes

- can be achieved with a complexity in $\mathcal{O}(m^2)$ multiplications over $\mathbb{F}_q$ and
- can correct up to $\frac{m_3+1}{2}$ errors.

These results allow to quantify the fault detection/correction capability of our scheme (where Tab. 3 simply showed the possibility to detect/correct).

Notice that the syndrome computation does not leak, if applied as per Eqn. 7, presented earlier in Sec. 4.

9 Conclusion

Code-based masking has recently allowed for efficient masking computations, for instance by allowing to share the computational load over multiple sensitive variables (e.g., the $m_1 = 16$ bytes making up the AES state). Those masking schemes have made judicious use of MDS codes in $\mathbb{F}_{256}$. In practice, these MDS constructions are Reed-Solomon codes, typically chosen from a Vandermonde generator matrix.

We observed that the masked multiplication algorithms could be enhanced by more stringent requirements on the codes. Namely, when the codes feature an orthonormal basis, numerical simplifications allow to reduce the number of field multiplications in the masked computations. In practice, we exhibit a computation with generalized Reed-Solomon codes; they are a superset of Reed-Solomon codes which offer more flexibility on their parameters and are such that some happen to admit an orthonormal basis. Thereby, we manage, for a given security level, to further reduce the complexity, in terms of time, memory and randomness requirement. As an important byproduct, our representation of masked data enables error detection and correction at any stage outside of masked addition/multiplication algorithms, without leaking sensitive information.

Acknowledgement. We thank Patrick Solé for his suggestion to construct orthonormal codes from self-dual codes.

A Appendix: Example of orthonormal codes

In this example we consider the finite field $\mathbb{F}_{2^8} = \frac{\mathbf{F}_2[x]}{(x^8+x^4+x^3+x+1)}$ used for AES, with $\alpha = \bar{x}$. Each element of $\mathbb{F}_{2^8}$ is represented by its numerical representation (*i.e.*, $\sum_{i=0}^{7} c_i \alpha^i$ is represented by the hexadecimal value of $(c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0)_2$). The listing below has been obtained using SAGE.

```
input: m = 5

C  =  GRS [4m, m, 3m+1] = [20, 5, 16] Reed-Solomon Code over GF(256)

C generator matrix G =
             01 00 00 00 00 41 03 3F AA 4E 19 07 18 0F F2 25 29 A2 91 8C
             00 01 00 00 00 72 DC 80 C3 BF C6 40 D3 27 2B 1F 8E 83 A0 EE
             00 00 01 00 00 9E CD 7E 11 EA 88 97 0E 65 02 05 9B 4E 84 D0
             00 00 00 01 00 B4 FC 77 44 50 AC D5 71 47 75 5A 70 FC 57 0F
             00 00 00 00 01 18 EF B7 3D 4A FA 04 B5 0B AF 64 4D 92 E3 BC

D =  GRS[4m, 2m-1, 2m+2] ⊥  = [20, 9, 12] ⊥  Reed-Solomon Code over GF(256)

D generator matrix  G* =
             01 00 00 00 00 00 00 00 00 00 00 E1 EA 17 2D 85 FC 51 BF A7
             00 01 00 00 00 00 00 00 00 00 00 B2 55 E8 DF 83 8D 47 D5 4D
             00 00 01 00 00 00 00 00 00 00 00 27 AD CB AA DB 65 47 29 3A
             00 00 00 01 00 00 00 00 00 00 00 2A 75 B8 3C CB FC C4 36 1F
             00 00 00 00 01 00 00 00 00 00 00 7B 44 C5 C7 B2 56 F4 2E 02
             00 00 00 00 00 01 00 00 00 00 00 6E 3C 73 4C F4 B1 83 AB 01
             00 00 00 00 00 00 01 00 00 00 00 37 C0 AA BA D8 78 56 21 31
             00 00 00 00 00 00 00 01 00 00 00 D0 8B A8 0F B0 7C 92 BD 1E
             00 00 00 00 00 00 00 00 01 00 00 4C 10 AD 1E 15 A1 93 DA 13
```

```
            00 00 00 00 00 00 00 00 00 01 00 04 AB 12 C9 3A 82 61 A0 0C
            00 00 00 00 00 00 00 00 00 00 01 7F 8D 19 09 63 30 4A B8 42

v in D such that Hamming weight(v) = 2m

            01 00 00 00 00 00 00 00 00 00 00 E1 EA 17 2D 85 FC 51 BF A7

w =
            01 00 00 00 00 00 00 00 00 00 00 5D 49 FD 29 92 B5 0D 46 AD

H = MDS self-dual matrix
            01 88 82 6C 4A 31 3B 7B 2A 76
            00 AA 5B 57 7F CC 13 CD 78 B3
            00 60 D3 89 52 EC E1 2B 63 2D
            00 70 8B FA D2 81 56 80 F5 71
            00 6F C8 B5 9C 02 2A 10 82 34

 H* = H systematic form
            01 00 00 00 00 33 C4 20 F2 24
            00 01 00 00 00 A2 E6 95 86 56
            00 00 01 00 00 27 A9 68 AD 4A
            00 00 00 01 00 71 BE 1F F8 29
            00 00 00 00 01 C6 34 C3 20 10

E =         33 C4 20 F2 24
            A2 E6 95 86 56
            27 A9 68 AD 4A
            71 BE 1F F8 29
            C6 34 C3 20 10

E x E ⊤  =
            01 00 00 00 00
            00 01 00 00 00
            00 00 01 00 00
            00 00 00 01 00
            00 00 00 00 01
```

References

[ABCV17] Sabine Azzi, Bruno Barras, Maria Christofi, and David Vigilant. Using linear codes as a fault countermeasure for nonlinear operations: application to AES and formal verification. *J. Cryptogr. Eng.*, 7(1):75–85, 2017.

[AK97] Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 1997.

[BBKN12] Alessandro Barenghi, Luca Breveglieri, Israel Koren, and David Naccache. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proc. IEEE*, 100(11):3056–3076, 2012.

[BCC+14] 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.

[BDL01] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptol.*, 14(2):101–119, 2001.

[BFG15] 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.

[BGHR14] Nicolas Bruneau, Sylvain Guilley, Annelie Heuser, and Olivier Rioul. Masks will fall off - higher-order optimal distinguishers. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 344–365. Springer, 2014.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 252–263. Springer, 2000.

[CDGT19] Claude Carlet, Abderrahman Daif, Sylvain Guilley, and Cédric Tavernier. Polynomial direct sum masking to protect against both SCA and FIA. *J. Cryptogr. Eng.*, 9(3):303–312, 2019.

[CGC+21] Wei Cheng, Sylvain Guilley, Claude Carlet, Sihem Mesnager, and Jean-Luc Danger. Optimizing inner product masking scheme by a coding theory approach. *IEEE Trans. Inf. Forensics Secur.*, 16:220–235, 2021.

[CGM19] Claude Carlet, Sylvain Guilley, and Sihem Mesnager. Direct sum masking as a countermeasure to side-channel and fault injection attacks. In José Luis Hernández Ramos and Antonio F. Skarmeta, editors, *Security and Privacy in the Internet of Things: Challenges and Solutions*, volume 27 of *Ambient Intelligence and Smart Environments*, pages 148–166. IOS Press, 2019.

[CGP+12] Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-order masking schemes for s-boxes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 366–384. Springer, 2012.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[DEG+18] Christoph Dobraunig, Maria Eichlseder, Hannes Groß, Stefan Mangard, Florian Mendel, and Robert Primas. Statistical ineffective fault attacks on masked AES with fault countermeasures. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 315–342. Springer, 2018.

[GG08] Markus Grassl and T. Aaron Gulliver. On self-dual MDS codes. In Frank R. Kschischang and En-Hui Yang, editors, *2008 IEEE International Symposium on Information Theory, ISIT 2008, Toronto, ON, Canada, July 6-11, 2008*, pages 1954–1957. IEEE, 2008.

[GM11] Louis Goubin and Ange Martinelli. Protecting AES with shamir's secret sharing scheme. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 79–94. Springer, 2011.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware* *and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Trans. Inf. Theory*, 45(6):1757–1767, 1999.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[JT12] Marc Joye and Michael Tunstall, editors. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012. ISBN: 978-3-642-29655-0; DOI: 10.1007/978-3-642-29656-7.

[KB10] Sabine Kampf and Martin Bossert. The euclidean algorithm for generalized minimum distance decoding of reed-solomon codes. In Marcus Greferath, Joachim Rosenthal, Alexander Barg, and Gilles Zémor, editors, *2010 IEEE Information Theory Workshop, ITW 2010, Dublin, Ireland, August 30 - September 3, 2010*, pages 1–5. IEEE, 2010.

[KL04] Jon-Lark Kim and Yoonjin Lee. Euclidean and hermitian self-dual MDS codes over large finite fields. *J. Comb. Theory, Ser. A*, 105(1):79–95, 2004.

[LKO+21] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. Platypus: Software-based power side-channel attacks on x86. In *2021 2021 IEEE Symposium on Security and Privacy (SP)*, pages 355–371, Los Alamitos, CA, USA, may 2021. IEEE Computer Society.

[Mag12] Houssem Maghrebi. *Masking countermeasures against HO-DPA : security evaluation and enhancement by specific mask encodings. (Les contre-mesures par masquage contre les attaques HO-DPA : évaluation et amélioration de la sécurité en utilisant des encodages spécifiques)*. PhD thesis, Télécom ParisTech, France, 2012.

[McE77] Robert J. McEliece. Encyclopedia of mathematics and its applications. *The Theory of Information and Coding: A Mathematical Framework for Communication*, 1977.

[Mes00] Thomas S. Messerges. Using second-order power analysis to attack DPA resistant software. In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, volume 1965 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2000.

[MOG+20] Kit Murdock, David F. Oswald, Flavio D. Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. Plundervolt: Software-based fault injection attacks against intel SGX. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 1466–1482. IEEE, 2020.

[MRGD12] Houssem Maghrebi, Olivier Rioul, Sylvain Guilley, and Jean-Luc Danger. Comparison between side-channel analysis distinguishers. In Tat Wing Chim and Tsz Hon Yuen, editors, *Information and Communications Security - 14th International Conference, ICICS 2012, Hong Kong, China, October 29-31, 2012. Proceedings*, volume 7618 of *Lecture Notes in Computer Science*, pages 331–340. Springer, 2012.

[PGS+17] 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.

[PR11] Emmanuel Prouff and Thomas Roche. Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*, pages 63–78. Springer, 2011.

[Pub01] NIST FIPS Pub. 197: Advanced encryption standard (aes), federal information processing standards publication 197, us department of commerce/nist, november 26, 2001, 2001.

[RMB15] Chester Rebeiro, Debdeep Mukhopadhyay, and Sarani Bhattacharya. Springer, 2015.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427. Springer, 2010.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.

[RR86] Welch Lloyd R and Berlekamp Elwyn R. Error correction for algebraic block codes, December 1986. US Patent 4,633,470.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Sha07] Priti Shankar. Decoding reed-solomon codes using euclid's algorithm. *Resonance*, 12(4):37–51, 2007.

[WCG+22] Qianmei Wu, Wei Cheng, Sylvain Guilley, Fan Zhang, and Wei Fu. On efficient and secure code-based masking: A pragmatic evaluation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):192–222, 2022.

[WMCS20] Weijia Wang, Pierrick Méaux, Gaëtan Cassiers, and François-Xavier Standaert. Efficient and private computations with code-based masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):128–171, 2020.

[YJ00] Sung-Ming Yen and Marc Joye. Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis. *IEEE Trans. Computers*, 49(9):967–970, 2000. DOI: 10.1109/12.869328.