Quasi-linear masking to protect against both SCA and FIA

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

Abstract

The implementation of cryptographic algorithms must be protected against physical attacks. Side-channel and fault injection analyses are two prominent such implementation-level attacks. Protections against either do exist; they are characterized by security orders: the higher the order, the more difficult the attack.

In this paper, we leverage fast discrete Fourier transform to reduce the complexity of high-order masking, and extend it to allow for fault detection and/or correction. The security paradigm is that of code-based masking. Coding theory is amenable both to mix the information and masking material at a prescribed order, and to detect and/or correct errors purposely injected by an attacker. For the first time, we show that quasi-linear masking (pioneered by Goudarzi, Joux and Rivain at ASIACRYPT 2018) can be achieved alongside with cost amortisation. This technique consists in masking several symbols/bytes with the same masking material, therefore improving the efficiency of the masking. Similarly, it allows to optimize the detection capability of codes as linear codes are all the more efficient as the information to protect is longer.

Namely, we prove mathematically that our scheme features side-channel security order of $d+1-t$, detects d faults and corrects $\lfloor (d-1)/2 \rfloor$ faults, where $2d+1$ is the encoding length and t is the information size ($t \geq 1$). Applied to AES, one can get side-channel protection of order $d = 7$ when masking one column/line ($t = 4$ bytes) at once.

In addition to the theory, that makes use of the Frobenius Additive Fast Fourier Transform, we show performance results, both in software and hardware.

Keywords: Side-channel analysis (SCA), Fault injection analysis (FIA), Code-Based Masking (CBM), Fault Detection, Frobenius Additive Fast Fourier Transform (FAFFT), Cost amortization.

[*] University of Bergen, 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.

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

[¶] Secure-IC S.A.S. Rennes, France.

[‖] Hensoldt France, Plaisir, France.

1 Introduction

In this article we are interested in the security of block ciphers, such as the AES. Such algorithms encrypt and decrypt data using a key, which must remain secret. Nonetheless, the implementation of cryptographic algorithms is subject to several attacks, amongst which side-channel and fault injection attacks are especially powerful. Side-channel attacks consist in correlating guessed (key-dependent) variables with some information leakage, whereas fault injection attacks consist in correlating sensitive variables with the fault outcomes. Both attacks try exhaustively all values of a subkey, and carry out a sufficient amount of attacks so as to rebuild the complete key with a divide-and-conquer approach.

It is therefore paramount to protect implementations against those attacks. The protection against side-channel analysis is often based on "masking": it consists in computing with randomized intermediate variables in order to provably deter attempts from an attacker to correlate on the randomized leakage. The protection against fault injection can typically rely on provable mathematical techniques, such as error detection codes.

Recently, the "code-based masking" (CBM) paradigm has been introduced: it leverages codes to achieve protection against the two threats at the same time. A pair of complementary codes allows to linearly combine sensitive information with digital random numbers in such a way the randomness has maximal decorrelation power whilst ensuring the demasking remains possible at all times. The ability to handle faults is based on redundancy kept by codes, ensuring their length is large enough to enable a detection or correction capability meeting the requirements in terms of fault injection attacks coverage.

1.1 Background on masking

Masking, from a historical perspective. A consensual protection against side-channel analyses consists in randomizing data representation and computations. This method is commonly referred to as masking. Several masking schemes have been proposed already.

Let us recap briefly the different milestones this technique has passed over the years. First of all, a proof-of-concept leveraging data randomization has been introduced by the seminal work of Kocher et al. [KJJ99]. Some early implementations have been proposed, and it has soon become clear that high-order attacks could defeat lower order masking schemes. Hence the research for provable protections against higher-order attacks. Formal definitions have been put forward by Blömer et al. in [BGK04]. A constructive scheme has been proposed by Ishai et al. [ISW03] on bits. This scheme has been subsequently extended to words (e.g., bytes) by Rivain and Prouff [RP10]. Some tools to perform automatic proofs for such schemes have been developed, for instance by Barthe et al. [BBD+15].

Minimizing the number of multiplications. The bottleneck in terms of performance is the number of nonlinear multiplications (that is, multiplications of x by an element different from a linear combination of powers of x whose exponents are of the form $2^j - 1$), since the addition and linear multiplications pose no problem and all S-boxes over finite fields being polynomial, the global complexity of masking directly depends on the number of nonlinear multiplications in the unprotected algorithm.

Then, a great deal of research has been devoted to reducing the number of multiplications in cryptographic operations, as for instance [CPRR15]. According to the state-of-the-art before 2020, it seemed difficult to mask one element of the field $\mathbb{F}_{q^m}$ in a way ensuring a d^{th}-order probing security, with a better complexity than $\mathcal{O}(d^2)$ multiplications over $\mathbb{F}_{q^m}$.

Cost amortization and fault detection capability. In order to get the most from masking schemes, from a performance standpoint, some attempts have been made. One direction has been the simultaneous masking of several bytes, referred to as "cost amortization", as demonstrated constructively by Wang et al. [WMCS20]. Formerly, the same idea has been applied in the field of multiparty computation, under the name of "packed secret sharing" [DIK10]. It has required to make a difference between the number of shares (n) and the masking order (d). Moreover, our masking is compatible with builtin fault detection capability, tightly intertwined with the CBM design.

Quasi-linear masking complexity. Another direction for reducing the cost due to multiplications is in reducing the cost of each multiplication by leveraging spectral representations, such as the *Number Theoretic Transform* (NTT) as put forward first by Goudarzi, Joux and Rivain (GJR [GJR18]). Unfortunately the NTT works only for prime fields with odd characteristics which is not convenient in practice. Recently, the authors of [GPRV21] extended the GJR scheme of [GJR18] to the even characteristic by replacing the NTT by a Discrete Fourier Transform (abridged "DFT" in the sequel), namely the additive fast Fourier transform of Gao et al. [GM10]. The novel masking scheme is dubbed "GJR+".

The initial proposal of [GJR18] (GJR) and the modification of [GPRV21] (GJR+) considerably improved upon the state-of-the-art, since they allowed to reduce the complexity of multiplications from quadratic ($\mathcal{O}(d^2)$) to quasi-linear ($\mathcal{O}(d \log d)$). This improvement is significant because the multiplication is the bottleneck in terms of computational complexity.

But the "DFT" in general (and NTT in particular) have a drawback: the linear operations are no longer transparent. Instead of being linear in the number of shares (each share being applied the linear transformation on itself, individually), a quasi-linear operation shall be applied. Still, the overall complexity remains quasi-linear.

Code-Based Masking (CBM). Besides, CBM has been introduced as a new paradigm to capture the security properties of masking. It describes the masking scheme as the (space-vector) sum of an encoded information taken from a code C, with an encoded mask taken from a code D, that is "disjoint" from C. The main advantage of CBM is that the security order is simple to determine: namely, the masking order is equal to the dual distance of the masking code minus the number one [PGS+17]. Computing in CBM, including multiplications, has been put forward in [WMCS20]. Advantageously, CBM has been proven relevant to describe the capability to detect faults on top of a masking scheme: indeed, when the two space-vectors C and D are in direct sum but such that $\dim(C) + \dim(D) < n$ where n is the length of C (or D), the information can be encoded in a redundant manner, enabling detection or even correction.

1.2 Analysis of the state-of-the-art

We analyze in this subsection the drawbacks of existing quasi-linear masking, in particular [GPRV21].

No cost amortization nor fault detection capability. Despite the advantages in terms of performance of quasi-linear masking ([GJR18] and [GPRV21] as well), the technique described in these papers does not unleash the full potential in terms of masking efficiency and fault attack protection. Regarding the efficiency, these papers do neither address how to encode multiple bytes of information in one go. Besides, these papers do not show how to correct errors (it would require to encode redundant information, as for instance put forward in [CCG+20]).

Non-practical masking order. It is hinted in [GPRV21] that their quasi-linear masking "improves the efficiency of the masked cipher for a masking order $n \geq 64$ for the MiMC block cipher and $n \geq 512$ for the AES" (as they write in at the end of the introduction, page 602). These masking orders are non-practical. Indeed, in real life, masking order is rather low, such as 1, 2 or maximum 3.

Complex implementation. Besides, the technique of [GPRV21] involves a randomized Fourier transform. This is an obvious limitation in terms of efficiency.

Abstract specification. In [GPRV21], the DFT is not instantiated, which limits the ability to compare with other schemes. As a side-effect, this negatively impacts the clarity of the security proof (which requires cumbersome hypotheses, such as leaving the DFT out of the scope of the security analysis).

1.3 Our contributions

In this paper, we introduce a practical masking scheme, with quasi-linear complexity.

Security proof based on codes properties. Our masking algorithm is described as a CBM. Therefore, not only side-channel security order is related to a dual distance, but also the capability to detect & correct faults is also related to codes minimum distance. Namely, we prove mathematically that our scheme features side-channel security order of $d+1-t$, detects d faults and corrects $\lfloor (d-1)/2 \rfloor$ faults, where $2d+1$ is the encoding length and t is the information size ($t \geq 1$).

Cost amortization. Our masking algorithm allows to mask jointly several bytes, based on a proof leveraging coding theory (within the CBM paradigm). Former works involving quasi-linear masking are only concerned by masking individual bytes. Notice that cost amortization also has an advantage in terms of the efficiency of fault detection capability.

Practical and efficient DFT. We thoroughly studied several DFT algorithms, and deploy an efficient one. It offers improved efficiency owing to optimization from a numeric standpoint. Namely, it relies on a sparse representation with small & simple coefficients (e.g., most often, "1"s). This DFT can be leveraged in the same time for the computation of the masking and the error detection.

Implementation and performance validation. We show that our quasi-linear masking is easy implementable, in software and in hardware. Namely, we provide performance characterization in C and VHDL languages. In particular, they support the effectiveness of cost amortization.

1.4 Outline

Preliminary notions are given in Sec. 2. They focus on DFT computation as they are the most complex operation in masking. We propose in Sec. 3 to consider an original DFT method proposed in [WZ88] which is particularly adapted to both software and hardware implementation. Indeed, the Gao and Cantor methods that we mentioned could give similar theoretical complexity but would require a huge effort of implementation in practice. We show in Sec. 4 how to extend this masking to the case of simultaneous protection of several symbols. We propose in a second phase, in Sec. 5 to correct or detect errors and erasures of any codeword present anywhere in the process of the ciphering algorithm. The security rationale is detailed in Sec. 6, where we provide a value for the side-channel security order and the fault injection resistance capabilities. Implementation in C and VHDL are given in Sec. 7, along with performance results. Conclusions are in Sec. 8.

Examples of quasi-linear DFT constructions adapted to handling bytes are given in App. A. We show the efficiency of this method on all platforms; our method definitively complies with hardware and software implementation and has a very low complexity. Namely, in App. A.1 (resp. App. A.2), we investigate the case of $d = 2$ (resp. $d = 7$).

2 Preliminaries

2.1 Finite fields

In this article, we are interested in data represented as elements from finite fields. We denote by $\mathbb{F}_q$ the field of q elements. We recall that when q is a power of two, $\mathbb{F}_q$ is said of characteristic two; in this case, subtraction and addition are the same operation, simply denoted by "+". A finite field of characteristic two can be seen as a polynomial extension of degree ℓ of $\mathbb{F}_2$, where $q = 2^\ell$. In this case, the addition boils down to the ℓ-bit parallel `XOR` operation. In this article, we illustrate our results on $\mathbb{F}_{256}$ (i.e., $\ell = 8$), which is the natural field within AES. Let ν be a primitive element of $\mathbb{F}_q$, that is a generator of the multiplicative group $\mathbb{F}_q^*$. Let n a positive integer. We assume that n divides $q - 1$, then we have that the field element $\omega = \nu^{\frac{q-1}{n}}$ is a primitive root of the unity. That is, $\omega^n = 1$. By construction, n is odd with q is power of two. We denote $n = 2d+1$.

2.2 Reed-Solomon codes

We denote by $\mathbb{F}_q^n$ the vector space of n field elements. A vector subspace of $\mathbb{F}_q^n$ is also called a linear code of length n. The Reed-Solomon code of length n, dimension k and minimal distance $n - k + 1$ is an evaluation code for which a generator matrix which can be defined as the evaluation of the polynomial basis $1, X, X^2, \ldots, X^{k-1}$ over the set $1, \omega, \omega^2, \ldots, \omega^{n-1}$. We denote this code by $\mathrm{RS}[n, k, n-k+1]$.

The dual $C^\perp$ of a linear code C is the linear code equal to the kernel of the generator matrix of C. It is well-known that the dual code of $\mathrm{RS}[n, k, n-k+1]$ is a $\mathrm{RS}[n, n-k, k+1]$ code.

As a consequence, we know that the matrix $(\omega^{ij})_{0 \le i \le k-1, 0 \le j \le n-1}$, known as the Vandermonde matrix defined over $1, \omega, \omega^2, \ldots, \omega^{n-1}$, is a generator matrix of the $\mathrm{RS}[n, n, 1]$ code. We have also that the inverse of the Vandermonde matrix corresponds to the generator matrix of the $\mathrm{RS}[n, n, 1]$ code defined over $1, \omega^{n-1}, \omega^{n-2}, \ldots, \omega^1$.

2.3 Multiplication of polynomials and DFTs in finite fields

We are interested in the multiplication of two polynomials P and Q on $\mathbb{F}_q$ of degree less or equal to d. The result is PQ, a polynomial of degree less or equal to $2d$.

The naïve computation has complexity $\mathcal{O}(d^2)$. However, a less complex method can be implemented.

Every polynomial is evaluated over its support (that is $\{1, \omega, \ldots, \omega^{n-1}\}$). The evaluation of PQ is the pairwise product of the evaluation of P and Q.

Thus, PQ is given by the interpolation of its truth table.

Now, it is well-known that the evaluation of a polynomial is precisely its Discrete Fourier Transform (DFT). Reciprocally, the interpolation of a polynomial is given by the inverse DFT (IDFT) [Knu11, Vol 2]. Notice that the definition of the DFT (and of the IDFT) is relative to the value of ω. Whenever there can be ambiguity, we shall write DFT_ω (resp. IDFT_ω) instead of DFT (resp. IDFT).

Besides, the evaluation of polynomial P on its support is equivalent to multiplying the row $(p_0, p_1, \ldots, p_{d-1})$ made up of coefficients of $P = \sum_{i=0}^{d-1} p_i X^i$ by the Vandermonde matrix. Reciprocally, the interpolation of a polynomial P is given by the multiplication by the row $(P(1), P(\omega), \ldots, P(\omega^{n-1}))$ with the inverse of the Vandermonde matrix.

Thus, for any vector $(p_0, \ldots, p_{2d}) \in \mathbb{F}_q^{2d+1}$, we can associate the polynomial $P(X) = p_0 + p_1 X + \ldots + p_{2d} X^{2d}$ and the discrete Fourier transform is defined by:

$$\mathrm{DFT}(p_0, \ldots, p_{2d}) = \left(\sum_{i=0}^{2d} p_i \omega^{ij}\right)_{j \in \{0,\ldots,2d\}} = \left(P(\omega^j)\right)_{j \in \{0\ldots 2d\}}.$$

Then the DFT inverse is defined by:

$$\mathrm{IDFT}(P(1), \ldots, P(\omega^{2d})) = \left(\sum_{i=0}^{2d} P(\omega^i)\omega^{-ij}\right)_{j \in \{0,\ldots,2d\}} = (p_0, \ldots, p_{2d}).$$

According to [Gao03], these operations (DFT and IDFT) can be computed using $\mathcal{O}(n \log(n) \log\log(n))$ operations in $\mathbb{F}_q$ operations. The details of these algorithms can be found in Chapters 8-11 of [vzGG13]. In general, it is known that computing products with DFT is computationally efficient for large values of n (e.g., $n > 1,000$).

Multiplicative DFT (see [Gao03]). The usual DFT requires that its support (n points, named a_i) form a multiplicative group of order n, concretely, the polynomial $X^n + 1$ has n distinct roots in the underlying field. In this case we say that the field supports DFT, and we call such a DFT multiplicative. A multiplicative DFT has time complexity $\mathcal{O}(n \log(n))$ and can be implemented in parallel time $\mathcal{O}(\log(n))$, where the implicit constants are small. For such abovementioned fields, we can take $n + 1$ to be a power of 2 with $n|(q-1)$ and $a_1, \ldots, a_n$ to be all the roots of $X^n + 1$. Then a DFT and its inverse at these points can be computed using $\mathcal{O}(n \log(n))$ operations in $\mathbb{F}_q$. By using DFTs, polynomial multiplication and division can also be computed using $\mathcal{O}(n \log(n))$ operations. The implicit constants in all these running times are very small, so these algorithms are practical for $n \geq 256$.

Additive DFT (see [GM10]). Unfortunately multiplicative DFTs are not supported by many finite fields, especially fields of characteristic two which are preferred in practical implementation. Cantor [Can89] finds a way to use the additive structure of the underlying field to perform a DFT over a finite field of order p^m where m is a power of p. This method is generalized by von zur Gathen and Gerhard [vzGG96] to arbitrary m. Their additive DFTs (for $p = 2$) uses $\mathcal{O}(n \log^2 n)$ additions and $\mathcal{O}(n \log^2 n)$ multiplications in $\mathbb{F}_q$. For fields of characteristic two and for $n = 2^m$, Gao and Mateer [GM10] recently improved on Cantor's method. When m is a power of 2, the above time complexity can be improved to $\mathcal{O}(n \log(n) \log\log(n))$. For arbitrary m, there is an additive DFT using $\mathcal{O}(n \log^2(n))$ additions and $\mathcal{O}(n \log(n))$ multiplications in $\mathbb{F}_q$. These DFTs are highly parallel and can be implemented in parallel time $\mathcal{O}(\log^2(n))$.

2.4 Quasi-linear DFT in practice

All DFT methods presented and discussed in the previous section 2.3 can be implemented in a pragmatic manner. Namely, first, a polynomial decomposition binary tree is computed off-line, once for all. Second, for each invocation of DFT or IDFT, a butterfly algorithm is executed on the pre-computed tree.

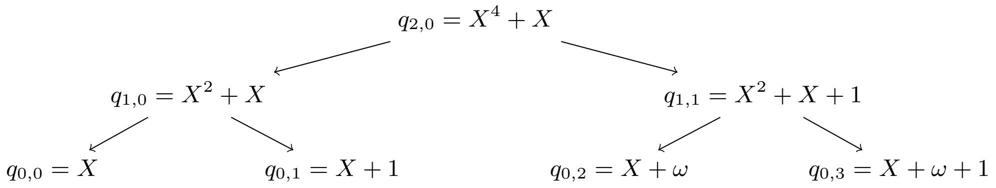
Preparation of a polynomial decomposition tree. We leverage the method put forward by Wang and Zhu in [WZ88]. Their idea consists in remarking that $P(\nu^i) = P(X) \bmod (X + \nu^i)$, then it is shown that the polynomial $X^{n+1} + X$ can be decomposed, as discussed below.

Let us design a binary tree of polynomials $q_{i,j}$, where i is the depth and j is an index for the breadth. Let n be the size of the DFT, then $0 \leq i \leq \lceil \log_2(n) \rceil$, and $0 \leq j \leq 2^{\lceil \log_2(n) \rceil - i}$. The tree is defined recursively as follows:

- The root is denoted by $q_{\lceil \log_2(n) \rceil, 0} = X^{n+1} + X$;
- intermediate nodes are denoted by $q_{i,j}$ and defined as $q_{i,j} = \prod_{k=0}^{1} q_{i-1,2j+k}$, with degree$(q_{i,j}) = 2^i$;
- Eventually, the leaves are $q_{0,j} = X - \beta_j$, where β_j are elements of $\mathbb{F}_q$.

By convention, the first leaf $q_{0,0} = 0$. In fact intermediate divisors are completely determined once the ordering of the bottom divisors $q_{i,0}$ is fixed.

Example 1 *We illustrate in this example such a binary tree, obtained from the Frobenius Additive Fast Fourier Transform (FAFFT) put forward in [LCK⁺18]. We remind that $X^4 + X = X(X+1)(X^2+X+1)$. The polynomial $X^2 + X + 1$ is the minimal polynomial whose zero is ω (recall that ω is defined throughout the article as a root of the unity of $X^n + 1$). Then we have the following binary tree:*



With the construction of [WZ88], it is possible to show that all $q_{i,j}$ are either linearized or affine polynomials [MS77] (that is: $q_{i,j}(X_1+X_2)+q_{i,j}(0) = q_{i,j}(X_1)+q_{i,j}(X_2)$). Consequently, polynomials $q_{i,j}$ are sparse with at most $i+1$ coefficients.

Computation of an efficient DFT. Based on such a pre-computed binary tree, we can now introduce an algorithm to efficiently compute the DFT. It is given in Alg. 1.

Algorithm 1: Quasi-linear (i.e., fast) Discrete Fourier Transform

```
Data: Pre-computed binary tree $q_{i,j}$
Input: $a = (a_0, a_1, \dots, a_{n-1})$
Output: $(b_0, b_1, \dots, b_{n-1})$ the DFT of $a$
$P_{\lceil \log_2(n) \rceil, 0} \leftarrow \sum_{i=0}^{n-1} a_i X^i$
for $i \in \{\lceil \log_2(n) \rceil - 1, \lceil \log_2(n) \rceil - 2, \dots, 0\}$ do
    for $j \in \{1, \dots, 2^{\lceil \log_2(n) \rceil - i}\}$ do
        $P_{i,j} \leftarrow P_{i+1, \lfloor j/2 \rfloor} \bmod q_{i,j}$
return $(P_{0,j})_{0 \leq j \leq n-1}) = (b_0, b_1, \dots, b_{n-1})$
```

The last step in Alg. 1 (for $i = 0$) consists in a reduction modulo $q_{0,j}$, which are polynomials of degree 1. Thus, the modulo operations yield a value in $\mathbb{F}_q$.

3 Quasi-Linear Masking without Cost Amortization

In this section, we introduce our high-order CBM algorithm, without cost amortization. That is, we consider only the masking of $t = 1$ element (byte). The purpose of this particular case is to explain simply the DFT-based masking with fault detection capability.

3.1 Masking construction

We define now the Reed-Solomon code $RS_q[n, n, 1]$ whose generator matrix is given by the Vandermonde Matrix $M \in \mathbb{F}_q^{n \times n}$ where $M_{i,j} = \omega^{ij}$. Let $x \in \mathbb{F}_q$ be a sensitive variable. To mask it, we pick randomly $r_0, \dots, r_{d-1}$ in $\mathbb{F}_q$ and encode the vector $\vec{a} = (x, r_0, \dots, r_{d-1}, 0, \dots, 0) \in \mathbb{F}_q^n$ with the Vandermonde matrix. We define:

$$\texttt{mask}(x) := \mathrm{DFT}(\vec{a}) = \left(\sum_{i=0}^{d} a_i \omega^{ij}\right)_{j \in \{0, \dots, 2d\}} = \vec{a} \cdot M \ .$$

Unmasking corresponds to the computation of the inverse DFT. Namely, let us denote $\vec{z} = \texttt{mask}(x)$ (i.e. $z_j = \sum_{i=0}^{d} a_i \omega^{ij}$). We have $\vec{a} = \mathrm{IDFT}(\vec{z})$. The sensitive data is $x = a_0$, thus we get:

$$\texttt{unmask}(\vec{z}) = \mathrm{IDFT}(\vec{z})_0 = \left(\vec{z} \cdot M^{-1}\right)_0 \ .$$

3.2 Masking addition and scaling

Let us denote: $\vec{z} = \texttt{mask}(x)$ and $\vec{z}\,' = \texttt{mask}(x')$. The following properties are satisfied:

- $\texttt{mask}(x + x') = \vec{z} + \vec{z}\,'$,
- $\texttt{mask}(\lambda x) = \lambda \cdot \vec{z} \quad$ for any $\lambda \in \mathbb{F}_q$.

3.3 Masking the multiplication

The multiplication is not a linear operation, so the question is how to compute $\texttt{mask}(xx')$ without unmasking x or x'. We denote $\vec{y} = \vec{z} * \vec{z}\,' := (z_j z'_j)_{j \in \{0,\ldots,2d\}}$ where "$*$" is the term-to-term product between two vectors. For $j \in \{1, \ldots, 2d\}$, we have:

$$\begin{aligned} & y_j &=& \ z_j z'_j = \left(x + \textstyle\sum_{i=1}^{d} r_i \omega^{ij}\right)\left(x' + \textstyle\sum_{i=1}^{d} r_i \omega^{ij}\right) = xx' + \textstyle\sum_{i=1}^{2d} r''_i \omega^{ij} \\ \implies & \vec{y} &=& \ \mathrm{DFT}(xx', r''_1, \ldots, r''_{2d}). \end{aligned}$$

The coefficients r''_i are obtained from the multiplication between $Z(X) = x + \sum_{i=1}^{d} r_i X^i$ and $Z'(X) = x' + \sum_{i=1}^{d} r'_i X^i$. The multiplication between $Z(X)$ and $Z'(X)$ of degree d gives a polynomial $Y(X) = xx' + \sum_{i=1}^{2d} r'' X^i$ of degree $2d$. Thus, to get $\texttt{mask}(xx')$ we need to eliminate the coefficients r''_i for $i \in [d + 1 \ldots 2d]$.

3.3.1 Extracting the last coefficients algorithm

We have:

$$\begin{aligned} & Y(X) &=& \ xx' + \textstyle\sum_{i=1}^{2d} r''_i X^i = xx' + \sum_{i=1}^{d} r''_i X^i + \sum_{i=d+1}^{2d} r''_i X^i \ . \\ \implies & \vec{y} &=& \ \mathrm{DFT}(xx', r''_1, \ldots, r''_d, 0, \ldots, 0) + \mathrm{DFT}(0, \ldots, 0, r''_{d+1}, \ldots, r''_{2d}) \ . \\ & &=& \ \texttt{mask}(xx') + DFT(0, \ldots, 0, r''_{d+1}, \ldots, r''_{2d}) \ . \\ \implies & \texttt{mask}(xx') &=& \ \vec{y} + DFT(0, \ldots, 0, r''_{d+1}, \ldots, r''_{2d}) \ . \end{aligned}$$

Now to construct $\mathrm{DFT}(0, \ldots, 0, r''_{d+1}, \ldots, r''_{2d})$ we must come back to the definition of IDFT. We remind that:

$$\mathrm{IDFT}(\vec{y}) = \left(\textstyle\sum_{i=0}^{2d} y_i \omega^{-ij}\right)_{j \in \{0,\ldots,2d\}} = (xx', r''_1, \ldots, r''_{2d}).$$

But in our case we are interested only by the coefficients r''_j for $j \geq d + 1$, thus we have to evaluate:

$$r''_j = \sum_{i=0}^{2d} y_i \omega^{-ij} \quad \text{with} \quad d + 1 \leq j \leq 2d.$$

For $0 \leq j \leq d-1$ we have:

$$\begin{aligned}
r''_{j+d+1} &= \textstyle\sum_{i=0}^{2d} y_i \omega^{-i(j+d+1)} \\
&= \textstyle\sum_{i=0}^{2d} \Big(y_i \omega^{-i(d+1)} \Big) \omega^{-ij} \\
&= \textstyle\sum_{i=0}^{d} \Big(y_i \omega^{-i(d+1)} \Big) \omega^{-ij} + \sum_{i=1}^{d} \Big(y_{i+d} \omega^{-(i+d)(d+1)} \Big) \omega^{-(i+d)j} \\
&= \textstyle\sum_{i=0}^{d} \Big(y_i \omega^{-i(d+1)} \Big) \omega^{-ij} + y_d \omega^{-d(d+1+j)} + \sum_{i=0}^{d} \Big(y_{i+d} \omega^{-(i+d)(d+1)} \Big) \omega^{-(i+d)j} \\
&= \textstyle\sum_{i=0}^{d} \Big(y_i \omega^{-i(d+1)} \Big) \omega^{-ij} + y_d \omega^{-d(d+1+j)} + \omega^{-dj} \sum_{i=0}^{d} \Big(y_{i+d} \omega^{-(i+d)(d+1)} \Big) \omega^{-ij} \\
&= \textstyle\sum_{i=0}^{d} \mu_i \omega^{-ij} + \nu_j + w_j \sum_{i=0}^{d} \lambda_i \omega^{-ij} \ .
\end{aligned}$$

where:

- $\vec{\mu} = (y_i \omega^{-i(d+1)})_{0 \leq i \leq d}$.
- $\vec{\theta} = (y_d \omega^{-d(d+1+j)})_{0 \leq j \leq d-1}$.
- $\vec{\lambda} = (y_{i+d} \omega^{-(i+d)(d+1)})_{0 \leq i \leq d}$.
- $w_j = \omega^{-dj}$.

Then we can calculate:

$$\vec{r}\,'' = \Big(r''_{d+1}, \ldots, r''_{2d} \Big) = \Big(\mathrm{IDFT}(\vec{\mu}, 0, \ldots, 0) + \vec{\theta} + \vec{w} * \mathrm{IDFT}(\vec{\lambda}, 0, \ldots, 0) \Big). \quad (1)$$

This computation is formalized as a routine in Alg. 2.

3.3.2 Algorithm for the masked multiplication

Finally we get:

$$\mathtt{mask}(xx') = \vec{y} + \mathrm{DFT}(0, \ldots, 0, \vec{r}\,'') \ .$$

This computation is summarized in Alg. 3.

A tedious calculation of the complexity of this algorithm in terms of the number of multiplications in $\mathbb{F}_q$ is given in Tab. 1.

We note that the calculation of $\mathrm{IDFT}(\vec{\mu}, 0, \ldots, 0)$ and $\mathrm{IDFT}(\vec{\lambda}, 0, \ldots, 0)$ leads to calculate half of a Discrete Fourier Transform, whose complexity is the same as a full-domain DFT, but with smaller constant factors.

In conclusion, the complexity of addition is linear, that of multiplication is quasi-linear. Besides, masking and demasking costs $n \log(n)$ multiplications [TL20] over $\mathbb{F}_q$, hence is quasi-linear as well. As a conclusion, all operations can be computed in quasi-linear complexity.

Algorithm 2: `extractLastCoefficients` Complexity: $2n(1 + \log(n))$

```
Input: a vector $\vec{y} \in \mathbb{F}_q^n$
Output: $\vec{r}\,'' \in \mathbb{F}_q^n$
$\vec{\mu} \leftarrow \vec{0} \in \mathbb{F}_q^n$
$\vec{\lambda} \leftarrow \vec{0} \in \mathbb{F}_q^n$
Let $\vec{\theta} \in \mathbb{F}_q^d$
for $0 \leq i \leq d$ do
    $\mu_i \leftarrow y_i M^{-1}_{i,d+1}$    // $M^{-1}$ is a pre-calculated matrix where $M^{-1}_{i,j} = \omega^{-ij}$
    $\lambda_i \leftarrow y_{i+d} M^{-1}_{(i+d),(d+1)}$
for $0 \leq i \leq d-1$ do
    $\theta_i \leftarrow y_d M^{-1}_{d,d+1+i}$
$\vec{\mu}\,' \leftarrow \mathrm{IDFT}(\vec{\mu})$
$\vec{\lambda}\,' \leftarrow \mathrm{IDFT}(\vec{\lambda})$
$\vec{r}\,'' = \vec{0} \in \mathbb{F}_q^n$
for $0 \leq i \leq d-1$ do
    $r"_{d+1+i} \leftarrow \mu'_i + \theta_i + M^{-1}_{d,i} * \lambda'_i$
return $\vec{r}\,''$    // Result of Eqn. (1)
```

Table 1: Complexity of operations involved in the masked multiplication

Variable	Cost
$\vec{y}$	n
$\vec{\mu}$	$d+1$
$\vec{\theta}$	d
$\vec{\lambda}$	$d+1$
$\vec{r}\,''$	$4(d+1)\log(n) + d + d$
$\mathtt{mask}(xx')$	$1 + 4d + 4(d+1)\log(n)$

Algorithm 3: `oneElementMultiplication` Complexity: $n(3 + 2\log(n))$

```
Input: two masked elements $\vec{z} = \mathtt{mask}(x), \vec{z}\,' = \mathtt{mask}(x') \in \mathbb{F}_q^n$
Output: $\mathtt{mask}(xx') \in \mathbb{F}_q^n$
$\vec{y} \in \mathbb{F}_q^n$
for $0 \leq i \leq n-1$ do
    $y_i \leftarrow z_i z'_i$
$\vec{r}\,'' = $ extractLastCoefficients($\vec{y}$) // Call to routine of Alg. 2
return $\vec{y} + \mathrm{DFT}(0, \ldots, 0, \vec{r}\,'')$
```

4 Quasi-linear Masking with Cost Amortization

Let us now extend our quasi-linear masking to several information elements (e.g., bytes) simultaneously. This allows to explore a tradeoff between side-channel order (namely $d+1-t$) and the amount of information processed simultaneously (namely t).

We propose then to translate this procedure in term of error correcting codes. We consider a free family $1, \alpha, \alpha^2, \ldots, \alpha^d$ of $\mathbb{F}_q^{d+1}$ where $\alpha^i \neq \omega$ for any $0 \leq i \leq d$. We want now to mask the vector $\vec{x} = (x_0, \ldots, x_{t-1}) \in \mathbb{F}_q^t$ with $1 \leq t < d$. (the case $t = 1$ has been addressed in previous section 3.)

4.1 Encoding procedure

First we pick randomly $\vec{r} = (r_t, r_{t+1}, \ldots, r_d)$ in $\mathbb{F}_q^{d+1-t}$. By Lagrange interpolation, there exists a vector $\vec{a} = (a_0, a_1, \ldots, a_d)$ and the associated polynomial $P_{\vec{x}}(X) = a_0 + a_1 X + \cdots + a_d X^d$ of degree at most d that satisfies $P_{\vec{x}}(\alpha^i) = x_i$ for $i \in \{0, \ldots, t-1\}$ and $P_{\vec{x}}(\alpha^i) = r_i$ for $i \in \{t, \ldots, d\}$.

Let us define the matrix $A \in \mathbb{F}_q^{(d+1)\times(d+1)}$, where $A_{i,j} = \alpha^{ij}$ for any i, j in $\{0, \ldots, d\}$. We have:

$$\vec{a} = (\vec{x} \mid \vec{r}) \times A^{-1} \ .$$

The second step of encoding consists in computing $\mathrm{DFT}_\omega(a_0, \ldots, a_d, 0, \ldots, 0)$. Thus finally:

$$\texttt{mask}(\vec{x}) = \mathrm{DFT}_\omega(a_0, \ldots, a_d, 0, \ldots, 0) = \mathrm{DFT}_\omega\left((\vec{x} \mid \vec{r}) \times \left[A^{-1}|0\right]\right) \ .$$

In this equation, $(\vec{x} \mid \vec{r})$ is the row obtained by the concatenation of row vectors $\vec{x}$ and $\vec{r}$, and $\left[A^{-1}|0\right]$ is the vertical concatenation of matrices A^{-1} and 0.

This method is a $\mathcal{O}((d+1)^2)$ complexity encoding procedure, but we can do better with the following one. We can construct $P(X) = P'(X) + P''(X)$ by first picking randomly the polynomial $P''(X) = a_t X^t + \cdots + a_d X^d$, then we evaluate $P'(X) = a_0 + a_1 X + \cdots + a_{t-1} X^{t-1}$ over $1, \alpha, \ldots, \alpha^{t-1}$ which cost $t(d-t)$ multiplications over $\mathbb{F}_q$.

We want now to construct $P'(X)$ which allows to solve the following linear system:

$$\begin{aligned}
\underbrace{\begin{bmatrix} a_0 & \ldots & a_{t-1} \end{bmatrix}}_{\vec{a}\ '} \times A' &= \begin{bmatrix} x_0 + P''(1) & \ldots & x_i + P''(\alpha^i) & \ldots & x_{t-1} + P''(\alpha^{t-1}) \end{bmatrix} \\
&= \vec{x} + \begin{bmatrix} P''(1) & \ldots & P''(\alpha^i) & \ldots & P''(\alpha^{t-1}) \end{bmatrix} \\
&= \vec{x} + \underbrace{\begin{bmatrix} a_t & \ldots & a_d \end{bmatrix}}_{\vec{a}\ ''} \times A'' \ = \ \vec{x} + \vec{a}\ '' \times A'' \ ,
\end{aligned}$$

where:

- $A' \in \mathbb{F}_q^{t\times t}$, and $A'_{i,j} = A_{i,j}$ for any $0 \leq i, j < t$;
- $A'' \in \mathbb{F}_q^{(d+1-t)\times t}$, $A''_{i,j} = A_{i+t,j}$ for any $0 \leq i < d+1-t$ and $0 \leq j < t$;

- $\vec{a}\,'' \in \mathbb{F}_q^{d+1-t}$ is a random vector.

Thus, the calculation of $\vec{a} = (\vec{a}\,' \mid \vec{a}\,'') = \big((\vec{x} + \vec{a}\,'' \times A'') \times A'^{-1} \mid \vec{a}\,''\big)$ costs $t(d+1)$ multiplications over $\mathbb{F}_q$ (we note that A'' and A'^{-1} may advantageously be pre-computed). Again, the second step of encoding consists in computing $\mathrm{DFT}_\omega(\ \vec{a} \mid \vec{0}\ )$ of complexity $\mathcal{O}(n\log(n))$.

The overall masking procedure is given in Alg. 4. Decoding procedure follows the same tracks: we use the inverse discrete Fourier transformation to get $\vec{a}$, then we have: $\vec{x} = \vec{a}\,' \times A' + \vec{a}\,'' \times A''$ which gives has the same complexity as the masking operation.

```
Algorithm 4: mask                    Complexity: $t(d+1) + n\log(n)$
    Input: a sensitive vector $\vec{x} \in \mathbb{F}_q^t$
    Output: mask($\vec{x}$) $\in \mathbb{F}_q^n$
    $\vec{a}\,'' = (a_t, a_{t+1}, \ldots, a_d) \xleftarrow{\$} \mathbb{F}_q^{d+1-t}$
    $\vec{a}\,' \leftarrow (\vec{x} + \vec{a}\,'' \times A'') \times A'^{-1}$
    return $\mathrm{DFT}_\omega(\vec{a}\,' \mid \vec{a}\,'' \mid \vec{0})$
```

The masking refresh allows to update the random part of the masked word, it consists of adding $\texttt{mask}(\vec{0})$, namely $\texttt{refresh}(\texttt{mask}(\vec{x})) = \texttt{mask}(\vec{x}) + \texttt{mask}(\vec{0}) = \texttt{mask}(\vec{x})$.

4.2 Masking the multiplication

Let us denote $\vec{z} = \texttt{mask}(\vec{x})$ and $\vec{z}\,' = \texttt{mask}(\vec{x}\,')$. Obviously,

$$\vec{z} * \vec{z}\,' = \mathrm{DFT}_\omega(a_0, \ldots, a_d, 0, \ldots, 0) * \mathrm{DFT}_\omega(a'_0, \ldots, a'_d, 0, \ldots, 0).$$ [1]

The polynomial obtained by performing $\mathrm{DFT}_\omega^{-1}(\mathrm{DFT}_\omega(P_{\vec{x}}) \times \mathrm{DFT}_\omega(P_{\vec{x}'})) = P_{\vec{x}}(X) \times P_{\vec{x}'}(X) = C(X)$ is a $2d$ degree polynomial, which satisfies $C(\alpha^i) = P_{\vec{x}}(\alpha^i) \times P_{\vec{x}'}(\alpha^i) = x_i x'_i$ for any i in $\{0, \ldots, t-1\}$.

Now we have to propose a method that associates a degree d polynomial $D(X)$ to $C(X)$. This polynomial must satisfy the same properties: $D(\alpha^i) = C(\alpha^i)$ for all $0 \le i \le t-1$.

The authors of [GJR18] proposed the following construction for $t = 1$:

$$\begin{aligned} D(X) &= c_0 + c_1 X + \ldots + c_d X^d + \alpha^d (c_{d+1} X + \ldots + c_{2d} X^d) \\ &= c_0 + (c_1 + \alpha^d c_{d+1}) X + \cdots + (c_d + \alpha^d c_{2d}) X^d \ . \end{aligned}$$

Obviously, in this case $D(\alpha) = C(\alpha) = x_1 x'_1$. We propose to generalize this construction. Let:

$$U_j(X) = \alpha^{jd} \frac{(X - \alpha) \cdots (X - \alpha^{j-1})(X - \alpha^{j+1}) \cdots (X - \alpha^t)}{(\alpha^j - \alpha) \cdots (\alpha^j - \alpha^{j-1})(\alpha^j - \alpha^{j+1}) \cdots (\alpha^j - \alpha^t)}.$$

[1] The '$*$' operation means the pairwise product.

Hence, by construction, $U_j(\alpha^j) = \alpha^{jd}$ and $U_j(\alpha^i) = 0$ for any i in $\{0, \ldots, t-1\} \backslash \{j\}$ and $\deg(U_j(X)) = t-1$. Then we set:

$$\begin{array}{rl} D(X) = c_0 + c_1 X + \cdots + c_d X^d & + \quad \sum_{j=1}^{t} U_j(X)(c_{d+1}X + \cdots + c_{2d-t+1}X^{d-t+1}) \\ & + \quad \sum_{j=1}^{t} U_j(X) \sum_{i=1}^{t-1} c_{2d-t+1+i}\alpha^{jd-jt+j+ij}. \end{array}$$

The degree d polynomial $D(X)$ satisfies $D(\alpha^i) = C(\alpha^i) = x_i x'_i$ of $i \in \{0, \ldots, t-1\}$.

In order to build efficiently $\mathrm{DFT}_\omega(D(X))$, let us write:

$$\begin{array}{rll} D(X) & = c_0 + c_1 X + \cdots + c_d X^d & + \quad (c_{d+1}X + \cdots + c_{2d-t+1}X^{d-t+1}) \sum_{j=1}^{t} U_j(X) \\ & & + \quad \sum_{i=1}^{t-1} c_{2d-t+1+i} \sum_{j=1}^{t} U_j(X)\alpha^{jd-jt+j+ij}. \end{array}$$

Thus:

$$\begin{array}{rcl} \mathrm{DFT}_\omega(D(X)) & = & \mathrm{DFT}_\omega(C(X)) \\ & + & \mathrm{DFT}_\omega(c_{d+1}X^{d+1} + \cdots + c_{2d}X^{2d}) \\ & + & \mathrm{DFT}_\omega(c_{d+1}X + \cdots + c_{2d-t+1}X^{d-t+1}) * \vec{u} \\ & + & \sum_{i=1}^{t-1} c_{2d-t+1+i} \cdot G_i \\ & = & \mathtt{mask}(\vec{x} * \vec{x}\,') \end{array}$$

where $G_i = \mathrm{DFT}_\omega(\sum_{j=1}^{t} U_j(X)\alpha^{jd-jt+j+ij})$ for $i \in \{1, \ldots, t-1\}$ and $\vec{u} = \mathrm{DFT}_\omega(\sum_{j=1}^{t} U_j(X))$ are a pre-computed values, and $c_{d+1}, \ldots, c_{2d} = \mathtt{extractLastCoefficients}(\vec{z} * \vec{z}\,')$.

4.3 Matrix product masking

It is necessary to also define the matrix product operation, as this type of operations is essential to calculate `MixColumns` or `ShiftRows` for example, with $t \in \{4, 8, 16\}$. Let us denote by $L \in K^{t \times t}$ the public matrix, we need to construct an algorithm `MatrixProduct` such that:

$$\mathtt{MatrixProduct}(\mathtt{mask}(\vec{x}), L) = \mathtt{mask}(x \cdot L) \ .$$

Let us recall that the masking operation is a combination between 2 FFTs, that can be represented as a matrix product as follows:

$$\mathtt{mask}(\vec{x}) = (\vec{x}, \vec{r}, \vec{0}) \cdot N \qquad (2)$$

where:

$$N = \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix} \times M \in \mathbb{F}_q^{n \times n} \ .$$

Thus:

$$\mathtt{mask}(\vec{x} \cdot L) = \mathtt{mask}(\vec{x}) \cdot L' \ ,$$

where $L' = N^{-1} \times \begin{bmatrix} L & 0 \\ 0 & I \end{bmatrix} \times N$ is a pre-computed value. i.e:

$$\mathtt{MatrixProduct}(\mathtt{mask}(\vec{x}), L) = \mathtt{mask}(\vec{x}) \cdot L' \ .$$

```
Algorithm 5: severalByteProduct Complexity: $n(3 + t + 4\log(n))$
  Input: two vectors $\vec{z} = \texttt{mask}(\vec{x}) \in \mathbb{F}_q^n$ and $\vec{z}\,' = \texttt{mask}(\vec{x}\,') \in \mathbb{F}_q^n$
  Output: $\texttt{mask}(\vec{x} * \vec{x}\,') \in \mathbb{F}_q^n$
  $\vec{y} \in \mathbb{F}_q^n$
  for $i \in \{0, \dots, n-1\}$ do
      $y_i \leftarrow z_i z'_i$
  $\vec{c}\,'' = \texttt{extractLastCoefficients}(\vec{y}) = (c_{d+1}, \dots, c_{2d})$
  $\vec{c} \leftarrow (0, \dots, 0 \mid \vec{c}\,'') = (0, \dots, 0, c_{d+1}, \dots, c_{2d}) \in \mathbb{F}_q^n$.
  $\vec{v} \leftarrow \vec{0} \in \mathbb{F}_q^n$
  for $0 \le i < t - 1$ do
      for $0 \le j < n$ do
          $v_j \leftarrow v_j + G_{i+1,j} \cdot c_{2d-t+2+i}$
  $\vec{c}\,' \leftarrow \vec{0} \in \mathbb{F}_q^n$
  for $i \in \{1, \dots, d - t + 1\}$ do
      $c'_i \leftarrow c_{d+i}$
  $\vec{w}\,' \leftarrow \mathrm{DFT}(\vec{c}\,') \in \mathbb{F}_q^n$
  for $i \in \{0, \dots, n-1\}$ do
      $w'_i \leftarrow w_i u_i$
  return $\vec{y} + \mathrm{DFT}(\vec{c}) + \vec{w}\,' + \vec{v}$
```

4.4 Exponentiation algorithm

Let e be a power of 2, we denote $\vec{x}^{\,e} = (x_0^e, \dots, x_t^e) \in \mathbb{F}_q^{t+1}$. In order to calculate `SubBytes` transformation efficiently we need to calculate $\texttt{mask}(\vec{x}^{\,e})$. We have:

$$\begin{array}{rlcll} & \texttt{mask}(\vec{x})^e & = & (\vec{x}, \vec{r}, \vec{0})^e \cdot N^e & \text{(where } (N^e)_{i,j} = (N_{i,j})^e) \\ \Longrightarrow & \texttt{mask}(\vec{x})^e \cdot ((N^e)^{-1} \times N) & = & (\vec{x}, \vec{r}, \vec{0})^e \cdot N \ = \ \texttt{mask}(\vec{x}^e) \ . & \end{array}$$

The operations order in this case is very important, in fact the $\texttt{mask}(\vec{x})^e \cdot (N^e)^{-1}$ can divulge the sensitive data if it has been done like this, that's why it's mandatory to pre-compute $((N^e)^{-1} \times N)$ first, then calculate $\texttt{mask}(\vec{x})^e \cdot ((N^e)^{-1} \times N)$.

5 Detecting/correcting fault injections

5.1 Error correcting code interpretation

We note that by construction, there exists an invertible matrix R that satisfies:

$$\begin{pmatrix} a_0 \\ \vdots \\ a_{t-1} \\ a_t \\ \vdots \\ a_d \end{pmatrix} = R \times \begin{pmatrix} x_0 \\ \vdots \\ x_{t-1} \\ P(\alpha^t) \\ \vdots \\ P(\alpha^d) \end{pmatrix} .$$

We note that this DFT computation corresponds to the encoding in the Reed-Solomon code defined by the evaluation of $1, X, \ldots, X^d$ over $1, \omega, \omega^2, \ldots, \omega^{2d}$, and represented by a matrix V. Hence, we get that $\texttt{mask}(y) = yR^\top V$. We deduce that our masking algorithm corresponds to an encoding procedure with a generalized Reed-Solomon code of minimal distance $d+1$, dimension d and length $2d+1$.

5.2 Error detection method

We have seen previously that our masking technique corresponds to an encoding in $[n = 2d+1, k = d+1, d+1]$ Reed-Solomon code. We propose in this section to describe a known method based on syndrome decoding [Pet60, Mas69, Jr.65, BHP98] that does not leak sensitive information.

Our information on t words is included inside of $d+1$ words which are then encoded in the Reed-Solomon code of length $2d$. Next we assume that a reasonable number of faults is injected on this codeword c. This codeword is in correspondence with a degree $k-1 = d$ polynomials $c(X) = \mathrm{IDFT}_\omega(c)$ in $\mathbb{F}_q[X]$.

It corresponds to the classic problem of error correction in a noisy channel. The error can be interpreted as a vector $e = (e_0, e_1, \ldots, e_{n-1}) = \mathrm{DFT}_\omega^{-1}(e(X))$ with ϵ the non-zero coefficients (positions) from $\mathbb{F}_q$ and $e(X)$ is a degree $n-1$ polynomial over $\mathbb{F}_q$. Hence, we study the vector $y = c + e = (e_j)_{j \in [\![0,n-1]\!]}$.

To detect or correct the errors, we calculate a syndrome from y, which only depends on the error word e and not on the codeword c. We recall that the dual code of the $\mathrm{RS}[n,k]$ is the $\mathrm{RS}[n, n-k]$ code. A basis of this code is given by the monomials $1, X, \ldots, X^{n-k-1}$ which are evaluated over the set $1, \omega, \ldots, \omega^{n-1}$.

Proposition 1 (Fast syndrom evaluation) *Let $S = (S_0, S_1, \ldots, S_{n-k-1})$. It is a syndrome sequence which satisfies*

$$S = (S_j)_{j \in [\![0,n-k-1]\!]} = \left(\sum_{i=0}^{n-1} y_j \omega^{ij} \right)_{j \in [\![0,n-k-1]\!]} = DFT_\omega(y).$$

Since $\deg(c(X)) < k$*,* $S = DFT_\omega(y) = DFT_\omega(e)$ *which does not depend of* c*. We deduce that detecting the presence of faults injection (i.e. checking whether* $S \neq 0$*) can be computed in* $\mathcal{O}(n\log(n))$ *multiplications.*

To correct these faults, we need to construct the error locator *polynomial.* We introduce the vector $\lambda = (\lambda_j)_{j\in\llbracket 0,n-k-1\rrbracket}$ such that $\lambda_j = 0$ whenever the corresponding coefficient e_j of e is non-zero, and $\lambda_j \neq 0$, whenever $e_j = 0$. In this way, we have $\lambda_j \cdot e_j = 0$ for all $j \in \{0, \ldots, n-1\}$. If we denote $\Lambda(X) = \mathrm{DFT}_\omega(\lambda)$ and $E(X) = \mathrm{DFT}_\omega(e) = S$, then, due to the well-known convolution theorem of the DFT, we have

$$E(X)\Lambda(X) = 0 \bmod X^n - 1. \tag{3}$$

The ϵ roots $\omega^{-j_1}, \ldots, \omega^{-j_\epsilon}$ of the polynomial $\Lambda(X)$ correspond to the locations $j_1, \ldots, j_\epsilon$ of the erroneous positions in y. Therefore $\Lambda(X) = \Lambda_0 + \Lambda_1 X + \cdots + \Lambda_\epsilon X$ is called the "error locator polynomial".

Without losing in generality, $\Lambda(X)$ can be normalized by setting $\lambda_0 = 1$. Equation (3) gives rise to a linear system of n equations. From these equations, $n-k-t$ equations only depend on the $n-k$ coefficients from $E(X)$, which coincide with the elements $S_0, \ldots, S_{n-k-1}$ of the syndrome, and the unknown coefficients of the error locator polynomial $\lambda(X)$. Hence, we extract a linear system of $n-k-er$ equations and ϵ unknowns:

$$\underbrace{\begin{bmatrix} S_0 & S_1 & \ldots & S_{\epsilon-1} \\ & & \vdots & \\ S_i & S_{i+1} & \ldots & S_{\epsilon+i-1} \\ & & \vdots & \\ S_{n-k-er-1} & S_{n-k-\epsilon} & \ldots & S_{n-k-2} \end{bmatrix}}_{S} \times \underbrace{\begin{bmatrix} \Lambda_\epsilon \\ \vdots \\ \Lambda_i \\ \vdots \\ \Lambda_1 \end{bmatrix}}_{\Lambda} = \underbrace{\begin{bmatrix} -S_\epsilon \\ \vdots \\ -S_i \\ \vdots \\ -S_{n-k-1} \end{bmatrix}}_{T}. \tag{4}$$

Obviously, a unique solution can can exist as long as $\epsilon \leq \frac{n-k}{2}$ which means than we can correct not more than $\frac{n-k}{2} = \frac{d-1}{2}$ faults.

To avoid a large complexity to solve the system of equations (4), due to specific form of it, we can use the well-known Berlekamp-Massey algorithm that solves this system with a linear complexity.

At this point we have located the errors by constructing $\Lambda(X)$. Reconstructing the errors can be done by the Forney algorithm. It consists in calculating the error evaluator polynomial

$$\Omega(X) = Sp(X)\Lambda(X) \bmod 2er,$$

where $Sp(X)$ is the partial syndrome polynomial:

$$Sp(X) = s_0 + s_1X + s_2X^2 + \ldots + s_{2er-1}X^{2er-1}.$$

Finally the error is given by evaluating the quantity for $X_j = \omega^{i_j}$:

$$e_j = \frac{\Omega(X_j^{-1})}{\Lambda'(X_j^{-1})},$$

Table 2: Side-channel security order *versus* fault detection / correction

n	d	t	SCA order ($d + 1 - t$)	Nb. of detected faults	Nb. of corrected faults
5	2	1 2	2 1	2	0
15	7	1 2 $\vdots$ 7	7 6 $\vdots$ 1	7	3
17	8	1 2 $\vdots$ 8	8 7 $\vdots$ 1	8	3

where Λ' is the first derivative of Λ. These quantities can be again evaluated by using the DFT transform, hence correcting fault injection can be done with a linear complexity.

Exemplary tradeoffs are given in table 2.

5.3 Positive effect of cost amortization on fault detection capability

Let us fix a field $\mathbb{F}_q$ and a minimal distance d. Then, it is more efficient from the code length point of view to mask two (resp. $2k$) symbols together than each one (resp. each k) independently. Formally, let `BLLC` the `BestLengthLinearCode` function in Magma [Uni], which yields the minimal length of a code on $\mathbb{F}_q$ of a given dimension and minimum distance. We have that:

$$\mathtt{BLLC}(\mathbb{F}_q, 2 \times k, d) \leq 2 \times \mathtt{BLLC}(\mathbb{F}_q, k, d). \tag{5}$$

For instance, on $\mathbb{F}_{256}$, RS codes are minimum distance separable (MDS) and thus $\mathtt{BLLC}(\mathbb{F}_q, k, d) = k + d - 1$. Thus Eqn. (5) rewrites $2k + d - 1 \leq 2(k + d - 1) \iff d \geq 1$ which is always true.

6 Security proof

The security of our scheme depends of our encoding procedure, our gadget multiplication and our capacity to detect fault injection during the steps of computation of the cipher algorithm.

6.1 The encoding procedure

We remind that our encoding procedure of a vector $\vec{r} = (x_0, \dots, x_{t-1})$ has been defined in subsection 4.1 consists in picking randomly $\vec{r} = (r_t, r_{t+1}, \dots, r_d)$ in $\mathbb{F}_q^{d+1-t}$ and performing the operation:

$$\mathtt{mask}(\vec{x}) = DFT_\omega\left((\vec{x} \mid \vec{r}) \times \left[A^{-1}|0\right]\right).$$

We remind also that the matrix $A = (\alpha^{ij})_{i,j \in [\![0..d]\!]}$. A first approach is o show that our masking method corresponds to a special case of DSM scheme, then we propose to translate this operation in a *generic encoder* as defined in [WMCS20] (page 137, definition 13). Applying DFT_ω corresponds to Vandermonde matrix multiplication. This matrix corresponds to the generator matrix of the Reed-Solomon code RS$[n, n, 1]$ defined over $\mathbf{F}_q$. A generator matrix of this code is defined by the evaluation of the monomials $(X^i)_{i \in \{0,n-1\}}$ over $1, \omega, \dots, \omega^{n-1}$. The multiplication by $\left[A^{-1}|0\right]$ leads to cancel the last rows of the generator matrix of this RS$[n, n, 1]$ code which becomes a Reed Solomon code RS$[n, d+1, n-d]$. We denote R a generator matrix of this code. Hence,

$$\mathtt{mask}(\vec{x}) = (\vec{x}, \vec{r}) \times A^{-1} \times R$$

Remark 1 *Our first remark at this point it that $A^{-1} \times R$ is still a RS$[n, d+1, n-d]$ code that can detects $n-d-1$ errors. We propose consequently later in this section a method to detect errors without revealing information.*

We can rewrite our encoding procedure in

$$\mathtt{mask}(\vec{x}) = \left((\vec{x}, \vec{0}) \times A^{-1} \times R\right) \oplus \left((\vec{0}, \vec{r}) \times A^{-1} \times R\right) = \vec{x}G \oplus \vec{r}H,$$

where $G = (Id_t, 0)A^{-1}R$ and $H = (0, Id_{d+1-t})A^{-1}R$.

Proposition 2 *The masking operation* $\mathtt{mask}(\vec{x})$ *is a generic encoder.*

Proof 1 *We have seen that $\mathtt{mask}(\vec{x}) = \vec{x}G \oplus \vec{r}H$. By construction $rank(G) = t$ and $rank(H) = d+1-t$. If we denote $\mathcal{C}_G$ and $\mathcal{C}_H$ the codes respectively generated by the generator matrix G and H, then $\mathcal{C}_G \cap \mathcal{C}_H = \{0\}$. If we denote $B = \begin{pmatrix} G \\ H \end{pmatrix}$, then we have:*

$$\mathtt{mask}(\vec{x}) = (\vec{x}, \vec{r}) \times B$$

and the B satisfies the definition of a generic encoder denoted enc_B.

If we denote d' the minimal distance of $\mathcal{C}_H$: $d' = d(\mathcal{C}_H)$ then a direct consequence is that the encoding procedure enc_B is d'-private. Our task consists now in evaluating d'. We denote K the matrix which corresponds to the last $d+1-t$ rows of A^{-1}, then

$$H = (0, Id_{d+1-t})A^{-1}R = K \times R$$

where $R = RS[n, d+1, n-d]$. By construction, H is $(d+1-t) \times n$ matrix since $(0, Id_{d+1-t})A^{-1}$ is a full rank matrix.

It is well known that the parity check matrix of R that we can denote T is a Reed-Solomon code $RS[n, d, n-d+1]$ and we have $H^tT = 0$. Hence, $H^tT = K \times R \times {}^tT = 0$ and the subspace generated by the rows of T are included in the kernel of H. In fact we can upperbound d' by remarking that K is a full rank matrix, then $H = K \times R$ is $(d+1-t) \times n$ full rank matrix, thus $\ker(H)$ is a $(d+t) \times n$ full rank matrix and due to the Singleton bound, we know that the code generated by this matrix has a minimal distance upperbound by $d+2-t$. We denote $\mathcal{C}_{H^\perp}$ the code generated by $\ker(H)$, then d' is the minimal distance of $C_{H^\perp}$.

We can show that this bound can be reached in some cases. For example, if $K = (Id_{d+1-t}|0)$, then $K \times R$ corresponds to a generator matrix of a RS$[n, d+1-t, d+t+1]$ code for which the dual code is the RS$[n, d+t, d+2-t]$ code. Let β a primitive element of the field $\mathbf{F}_q$. We assume that $\alpha = \beta^{15}$ and $\omega = \beta^{17}$.

Study of K:

We remind that $K = (0, Id_{d+1-t})A^{-1}$. First of all, A^{-1} is a Reed-Solomon generator matrix as any invertible squarre matrix because it is equivalent up invertible matrix to a Reed-Solomon code. Hence K is a generator matrix of a sub code of a RS$[d+1, d+1]$ code. We would like to determine now the dual code of K and we observe the equation $A^{-1} \times A = Id_{d+1}$ which can be written

$$\begin{pmatrix} K'_{t\times(d+1)} \\ K_{(d+1-t)\times(d+1)} \end{pmatrix} \times \left(B_{(d+1)\times t}, B'_{(d+1)\times(d+1-t)} \right) = \begin{pmatrix} Id_t & 0_{t\times(d+1-t)} \\ 0_{(d+1-t)\times t} & Id_{d+1-t} \end{pmatrix}.$$

We deduce that $K_{(d+1-t)\times(d+1)} \times B_{(d+1)\times t} = 0_{(d+1-t)\times t}$ and we know that

$$K = K_{(d+1-t)\times(d+1)} \text{ and } B = Kernel(K) = B_{(d+1)\times t} = (\alpha^{ij})_{i\in[\![0..d]\!], j\in[\![0..t-1]\!]}.$$

By construction ${}^t(B_{(d+1)\times t})$ is a generator matrix of a RS$[d+1, t]$ code. We know that if $\mathcal{C}$ is a linear code, then $(C^\perp)^\perp = \mathcal{C}$, thus K is a generator matrix of a RS$[d+1, d+1-t]$ code, in particular its minimal distance is $d+2-t$.

We want now to describe the kernel of $K \times R$. We can repeat the same construction for R. If we denote V_ω the Vandermonde matrix associated to DFT_ω:

$$V_\omega \times V_\omega^{-1} = \begin{pmatrix} R_{(d+1)\times(2d+1)} \\ R'_{d\times(2d+1)} \end{pmatrix} \times \begin{pmatrix} Ri_{(2d+1)\times(d+1)}, Ri'_{(2d+1)\times d} \end{pmatrix}, \text{ and}$$

$$V_\omega \times V_\omega^{-1} = \begin{pmatrix} Id_{d+1} & 0_{(d+1)\times d} \\ 0_{d\times(d+1)} & Id_d \end{pmatrix}.$$

We deduce that $R_{(d+1)\times(2d+1)} \times Ri_{(2d+1)\times(d+1)} = Id_{d+1}$ with $R = R_{(d+1)\times(2d+1)}$. The matrix V_ω^{-1} is Vandermonde matrix associated to $IDFT_\omega$, then $R_i = Ri_{(2d+1)\times(d+1)} = (\omega^{-ij})_{i\in[\![0..2d]\!],j\in[\![0..d]\!]}$. We remark that $K \times R \times {}^tT = 0$ and $K \times R \times R_i \times B = K \times Id \times H = 0$. Hence we can build Vector space included in the kernel of $H = K \times R$ with T which is the generator matrix of a RS$[2d+1, d]$ code and $D = {}^tB \times {}^tR_i$.

We note that ${}^tR_i = (\omega^{(n-i)j})_{i\in[\![0..d]\!],j\in[\![0..2d]\!]}$ is a generator matrix of a code generated by $d+1$ polynomials of degree more than $d+1$. Then ${}^tB = (\alpha^{ij})_{i\in[\![0..t-1]\!],j\in[\![0..d]\!]}$. Hence the code generated by D is an evaluation code generated by t independant polynomials of degree more than $d+1$ whereas T is a generator matrix of a code generated by d polynomials of degree strictly less than d, then these two codes are linearly independant and we deduce that we have built the kernel of H. We have now to evaluate the minimal distance of this code. Thus, we get that

$$D = {}^tB \times {}^tRi = \left(\sum_{k=0}^{d} \alpha^{ik} \omega^{(2d+1-k)j} \right)_{i\in[\![0..t-1]\!],j\in[\![0..2d]\!]}.$$

Let

$$D_{i,j} = \sum_{k=0}^{d} \alpha^{ik} \omega^{(2d+1-k)j} = \omega^{(d+1)j} \sum_{k=0}^{d} \alpha^{ik} \omega^{(d-k)j}$$

and

$$D_{i,j} = \omega^{(d+1)j} \sum_{k=0}^{d} \alpha^{i(d-k)} \omega^{kj}.$$

Then

$$D_{i,j} = \alpha^{id}\omega^{(d+1)j} \sum_{k=0}^{d} \left(\frac{\omega^j}{\alpha^i}\right)^k = \alpha^{id}\omega^{(d+1)j} \frac{1-\left(\frac{\omega^j}{\alpha^i}\right)^{d+1}}{1-\frac{\omega^j}{\alpha^i}}.$$

If $i = 0$, then

$$D_{0,j} = \omega^{(d+1)j} \frac{1-\omega^{(d+1)j}}{1-\omega^j} = \frac{\omega^{(d+1)j} - \omega^j}{1-\omega^j} = \omega^j \frac{\omega^{jd}-1}{1-\omega^j}.$$

Hence

$$D_{0,j} = \omega^j + \omega^{2j} + \ldots + \omega^{(d-1)j} + \omega^{jd},$$

thus $(D_{0,j})_j$ corresponds to the evaluation of the polynomial $X + X^2 + \ldots + X^{d-1} + X^d$ on ω^j. In the same times T is a RS$[2d+1, d]$ Reed-Solomon code, thus $T \cup (D_{0,j})_j$ generates the RS$[2d+1, d+1]$ code.

We deduce here that if $t = 1$, the minimal distance of $T \cup D$ is $d' = d+1 = d+2-t$.

We showed that $D_{i,j} = \alpha^{id}\omega^{(d+1)j}\frac{1-\left(\frac{\omega^j}{\alpha^i}\right)^{d+1}}{1-\frac{\omega^j}{\alpha^i}} = \frac{\alpha^{i(d+1)}\omega^{(d+1)j}+\omega^j}{\alpha^i+\omega^j}$. It means that the vector D_i corresponds to the evaluation of the fraction

$$\frac{\alpha^{i(d+1)}X^{d+1}+X}{\alpha^i+X}$$

over $\{1,\omega,\dots,\omega^{2d}\}$ and we are looking for a degree d polynomial $P(X)$ that cancels the maximum of positions of D_i, i.e. such that $Q(X) = (X+\alpha^i)P(X) + X + \alpha^{i(d+1)}X^{d+1}$ admits the maximum of zeros. We remark that degree$(Q) \leq d+1$, then the number of zero is less than $d+1$ which is equivalent to a minimal distance greater than $2d+1-(d+1) = d$. Hence we deduce that for $t = 2$ $d' = d = d+2-t$.

for $t = 3$, we know that $d' \leq d+2-3 = d-1$, our previous proof shows that taking $D_{1,j}$ or $D_{2,j}$ leads to build a codeword of weigth greater than d, thus to expect better than d, we must combine these two vectors. Combining D_1 and D_2 means that for a fixed element $\theta \in \mathbf{F}_q$ we are looking for a degree d polynomial $P(X)$ such that for a maximum of input we have

$$P(X) = \frac{\alpha^{d+1}X^{d+1}+X}{\alpha+X} + \theta\frac{\alpha^{2(d+1)}X^{d+1}+X}{\alpha^2+X}.$$

This is equivalent of studying the number of zero of the function $T(X) = (X+\alpha)(X+\alpha^2)P(X) + (X+\alpha^2)(\alpha^{d+1}X^{d+1}+X) + \theta(X+\alpha)(\alpha^{2(d+1)}X^{d+1}+X)$. The degree of $T(X)$ is less or equal to $d+2$ then $T(X)$ has $d+2$ roots maximum and we deduce:

If $t = 3$, the minimal distance of $T \cup D$ is $d' = d-1 = d+2-t$.

By induction we have that for any t, $d' = d+2-t$ and the masquing order is $d+1-t$.

6.2 The multiplication gadget

The security of the masking *representation* is immediate owing to the number of shares. However, to be comprehensive, we have to show now that *operations* are also secure. Namely, the masked multiplication procedure offer also the same level of protection. The weakest side is obtained with the computation of

$$(c_{d+1},\dots,c_{2d}) = extractLastCoefficients(\vec{z} * \vec{z}').$$

with $P_{\vec{x}}(X) \times P_{\vec{x}'}(X) = C(X) = \sum_{i=0}^{2d} c_i X^i$ We can consider here that the adversary has access to the $d+1-t$ values from $c_{d+1},\dots,c_{2d}$. Hence the question is: *Does this knowledge provide information on the sensitive variables $\vec{x}$, $\vec{x}'$ and $\vec{x} * \vec{x}'$ according our previous notations?*

We remind that $\vec{z}$ and $\vec{z}'$ correspond respectively to the polynomials $P_{\vec{x}}(X) = \sum_{i=0}^{d} a_i X^i$ and $P_{\vec{x}'}(X) = \sum_{i=0}^{d} a'_i X^i$, with $(\vec{x},\vec{r}) = (P_{\vec{x}}(1), P_{\vec{x}}(\alpha),\dots,P_{\vec{x}}(2d))$

and $(\vec{x}', \vec{r}') = (P_{\vec{x}'}(1), P_{\vec{x}'}(\alpha), \ldots, P_{\vec{x}'}(2d))$. Hence, we remark that we have the following system of equations:

$$\begin{cases} c_{2d} & = & a_d a'_d \\ c_{2d-1} & = & a_{d-1}a'_d + a_d a'_{d-1} \\ c_{2d-2} & = & a_{d-2}a'_d + a'_{d-2}a_d + a'_{d-1}a_{d-1} \\ & \vdots & \\ c_{2d-k} & = & \sum_{i=0}^{k} a_{d-i}a'_{d-(k-i)} \\ & \vdots & \\ c_{d+1} & = & \sum_{i=0}^{d-1} a_{d-i}a'_{i+1} \end{cases}$$

The attacker has maximum $d+1-t$ equations from this system and we remind that $(a_0, a_1, \ldots, a_d) = (\vec{x}, \vec{r}) \times A^{-1}$. We have proven previously that this encoding procedure corresponds to a generic encoder with

$$(\vec{x}, \vec{r}) \times A^{-1} = \vec{x} \times (Id_t, 0)A^{-1} + \vec{r} \times (0, Id_{d+1-t})A^{-1}$$

with $K = (0, Id_{d+1-t})A^{-1}$ that has a parity check matrix of minimal distance $d+2-t$. This implies that the attacker must get at least $d+2-t$ values of a_i or a'_i to get some sensitive informations. Let's assume that the attacker has access to the complete system, then we can evaluate the number of potential solutions for $(a_i)_{i \in [\![d, 2d]\!]}$: we can assume that $c_{2d} \neq 0$, then the equation $c_{2d} = a_d a'_d$ admits $2^m - 1$ solutions. If $c_{2d} = 0$, then $a_d a'_d$ admits 2^m solutions. By setting $a_d \neq 0$ and $a'_d \neq 0$ we get the equation $c_{2d-1} = a_{d-1}a'_d + a_d a'_{d-1}$ admits 2^m solutions. By induction, we get the same property at any step $k \leq d-1$. Thus totally this system admits $2^{m(d-1)}$ solutions. This result must be worst with less equations, thus this system of equation does not give information on at least $d+2-t$ values of (a_i) solutions. We deduce that the gadget multiplication is $d+1-t$ secure probbing.

6.3 Fault detection/correction

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 minimal distance. The detection is more subtle in front of low weight faults, as the minimum distance of code is involved.

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 4.1, each masked element belongs to the code $\mathrm{RS}[n, d+1, n-d]$. 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 non-correctible 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 consider the problem of decoding Reed-Solomon codes, erasures can simply be considered as errors. Hence, a decoding algorithm that works for 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. 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] and as it is explained in the subsection 6.3. Hence, this algorithm returns the vector corresponding to the error, that allows to return the corrected codeword belonging to the Reed-Solomon code. Never the sensitive information has been 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 the previous subsection regarding the encoding procedure, we have seen that masking a vector $\vec{x}$ consists in performing

$$\mathtt{mask}(\vec{x}) = (\vec{x}, \vec{r}) \times A^{-1} \times R.$$

Hence $\vec{z} = \mathtt{mask}(\vec{x})$ is simply a codeword belonging to the $RS(n, d+1, d+1)$ code. If we denote V the parity check matrix of R, we have by construction $R \times V = 0$ and in particular $\mathtt{mask}(\vec{x}) \times V = 0$. 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:

$$\vec{\epsilon} = \vec{z}\,' \times V = \vec{z} \times V + \vec{e} \times V = \vec{e} \times V.$$

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 (to enable key zeroization for instance), or the number of errors is reasonable and the error correction algorithm can correct the disturbed multiplication.

Eventually, it is up to the security policy to consider the best strategy between detecting and launching a countermeasure or correcting.

7 Performances

The implementation of a masked AES allowed us to accurately measure the gain in time and memory space that can be obtained with parallel masking. Indeed, as we can see in Fig. 1, the computation time decreases linearly according to the size of the sensitive data (t). This figure shows the computation duration for the full AES and also the breakdown in its internal functions (`SubBytes`, `ShiftRows`, `MixColumns`, and `XOR` operations). This is mainly due to the size of the masked block which is reduced ($n \times 16/t$) as can be seen in Figure 2. Thus, reducing the overall size of the masked word made it possible to avoid more expensive operations such as multiplication and exponentiation.

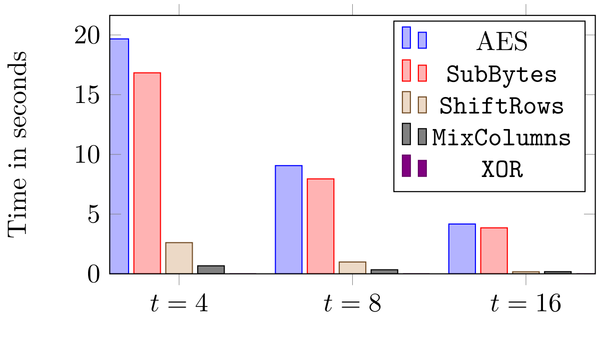


Figure 1: Computation time for 50 times AES calculation, for $(d, n) = (25, 51)$.

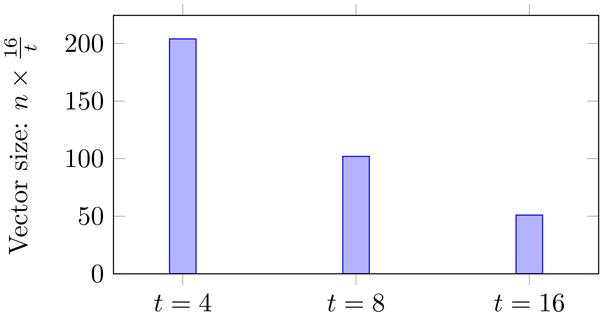


Figure 2: Masked value length for 16 bytes with parameters $(d, n) = (25, 51)$.

Table 3: Synthesis results for the case $d = 7$ in $\mathbb{F}_{256}$.

Perf. $\backslash$ Design	**Vandermonde**	**DFT involved in our masking**
Area (#gates)	2711	1136
Logic depth	11	11

7.1 Hardware

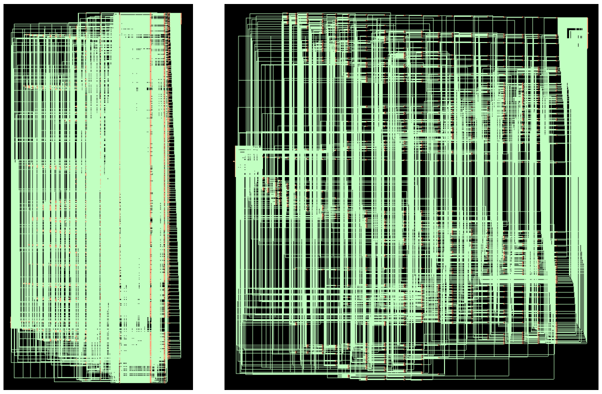
We compare the naïve Vandermonde method with our method based on FAFFT, for $d = 7$ (see Tab. 2). The polynomial decomposition tree we use is that given in appendix A.2.

For the evaluation, both designs are represented in VHDL. We use `genus` from Cadence as a Computed Aided Design (CAD) tool, in a simple library consisting only of two-input gates. The synthesis results are given in Tab. 3.

We notice that both implementations only consist only in exclusive-or (that is: `XOR`) gates. Indeed, we pushed the simplification to the maximal effort in the CAD tool which has yielded to the simplification of bitwise products (`AND` gates) with constants. These simplifications consist in constant propagation within the netlist[1].

Our implementation is fully combinational, hence a similar (actually the same) depth for both implementations. Given the regular structure of the polynomial decomposition tree, a sequential approach is possible, that enables a trade-off between time with less area. This can be seen from the netlist (see Fig. 3): our design yields a netlist which has almost constant width throughout its depth.

[1] Notice that article [LCK+18] also analyses gate-level complexity but somehow obtains a non-optimized netlist consisting of both `XOR` and `AND` gates.



(a) (b)

Figure 3: Netlist of (a) Vandermonde and (b) our fast DFT design.

8 Conclusions

Code-based masking (CBM) can implement arbitrary computations based on additions and multiplications, whist ensuring arbitrary chosen side-channel security order. Besides, in terms of complexity, it has already been shown that those operations can be carried out in quasi-linear time.

In this article, we show for the first time that such properties can be extended to the case of multiple bytes concomitant masking (construction known as leakage amortization). We also show how such masking is compatible with error detection and/or correction, that can be nested within the code-based masking representation.

Furthermore, we detail the computation of the required Discrete Fourier Transform (DFT) involved in these operations. We show how it can be implemented efficiently for some specific DFT algorithms, which have a small implementation-level complexity (e.g., in binary gates count).

We show actual implementation complexity results both in software and in hardware, and detail our gain both in terms of performance and of gate area.

A Case of the Galois field $\mathbb{F}_{2^8}$

The symmetric encryption algorithm AES is a byte-oriented block cipher. It design leverages the irreducible polynomial $X^8+X^4+X^3+X+1$. The Sbox is based on the inverse function defined over the finite field $\mathbb{F}_{2^8} = \frac{\mathbb{F}_2[X]}{(X^8+X^4+X^3+X+1)}$. The canonical basis is given by $\alpha = \overline{X}$ in $\mathbb{F}_{2^8}$ and $1+\alpha$ is a primitive element of this field. Then $X^{256} - X = X(X^{255} - 1)$ and $255 = 3 \times 5 \times 17$. We can consider DFT with $n = 3, 5, 15, 17, 51, 85, 255$. The case $n = 3$ has been described in a previous section.

We note that we have not a large choice for n if we keep this method. We will see in the next section that we can construct a DFT and its associate inverse by observing the different trees.

The source code of our implementations is provided in a GitHub: https://github.com/<anonymized_for_review>/FFT_masking.git.

A.1 AES example with $d = 2$

The case $n = 2d+1 = 5$ corresponds to $d+1-t = 3-t$ order masking maximum. The case $n = 5$ is not a power of two but we can propose a decomposition that lead to very fast complexity and we consider $\omega = (1+\alpha)^{\frac{255}{5}} = (1+\alpha)^{51}$, then

$$X^6 - X = X(X-1)(1+X+X^2+X^3+X^4) = X(X-1)(X-\omega)(X-\omega^2)(X-\omega^3)(X-\omega^4).$$

Hence, we can propose the polynomial decomposition tree displayed in Fig. 4.

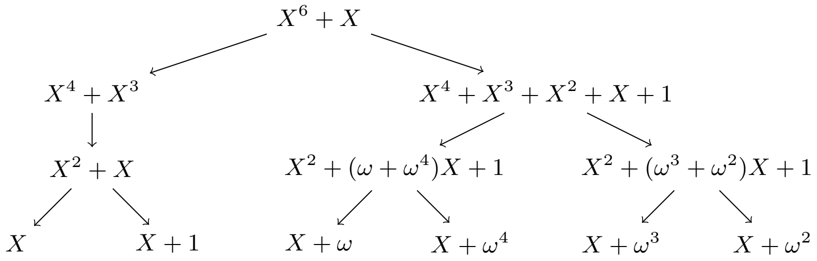


Figure 4: Polynomial decomposition tree for $X^6 + X$ on $\mathbb{F}_{256}$.

We propose to evaluate precisely here the complexity of the $\vec{r}\,''$ calculation with

$$\vec{r}\,'' = \Big(\mathrm{IDFT}(\vec{\mu}, 0, \ldots, 0) + \vec{\theta} + \vec{w} * \mathrm{IDFT}(\vec{\lambda}, 0, \ldots, 0)\Big).$$

Hence this computation leads to consider a maximum degree 3 polynomial $P(X)$ that we have to evaluate over $\{1, \omega, \ldots, \omega^4\}$.

The Euclidean division of $P(X)$ by $X^2 + X$ costs 2 additions over $\mathbb{F}_{2^8}$. The Euclidean division of $P(X)$ by $X^2 + (\omega + \omega^4)X + 1$ costs 2 multiplications and 4 additions over $\mathbb{F}_{2^8}$. We obviously get the same number for $X^2 + (\omega + \omega^4)X + 1$. The last step consists by performing the Euclidean division by all monomials except X which costs: 5 additions and 4 multiplications. Hence totally the DFT cost 6 multiplications and 9 additions. For comparison, $11 > 6\ln(6) > 10$ and $20 > 6\ln^2(6) > 19$. We deduce that

A.2 AES example with $d = 7$

The case $n = 2d + 1 = 15$ corresponds to $d + 1 - t = 8 - t$-order masking maximum. The case $n = 15$ corresponds to a power of two and we can propose

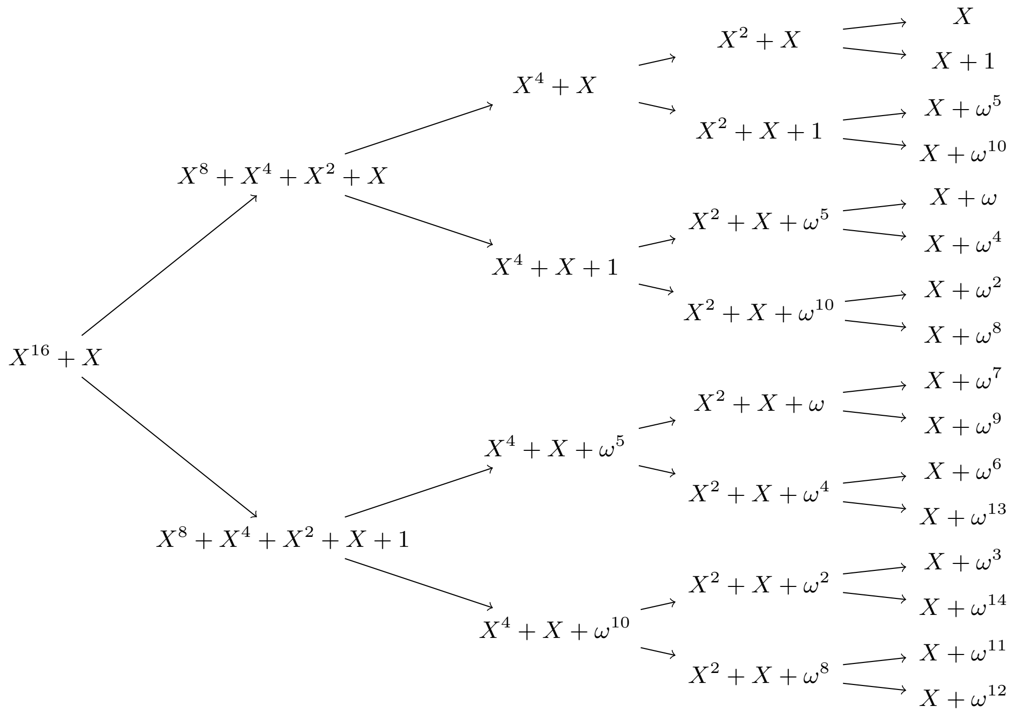


Figure 5: Polynomial decomposition tree for $X^{16}+X$ on $\mathbb{F}_{256}$.

a decomposition that lead to very fast complexity. Let $\omega = (1+\alpha)^{\frac{255}{15}} = (1+\alpha)^{17} = 1+\alpha^5+\alpha^6+\alpha^7$. Then we get that

$$\begin{aligned}
&1+\omega^2=\omega^8;\\
&1+\omega=\omega^4;\\
&1+\omega^7=\omega^9;\\
&1+\omega^3=\omega^{14};\\
&1+\omega^5=\omega^{10};\\
&1+\omega^{11}=\omega^{12};
\end{aligned}$$

thus, according to [WZ88], we get the following decomposition tree depicted in Fig. 5. This tree is rotated so that it fits in the page limits.

Regarding AES, block size is 16 bytes then we can apply three Fourier transforms over respectively 5 bytes, 6 bytes and 5 bytes. It means that we encode polynomials of degree at most 7. Hence in the diagram, we start from evaluating the division by a degree 4 polynomials.

A.2.1 Calculation modulo degree 8 polynomials

Let $a(x) = a_{15}x^{15}+a_{14}x^{14}+\ldots+a_1x+a_0$, then $a(x) = (x^8+x^4+x^2+x)q(x)+r(x)$ with

$$\begin{aligned} q(x) &= a_{15}x^7 + a_{14}x^6 + a_{13}x^5 + a_{12}x^4 + (a_{11} + a_{15})x^3 + (a_{10} + a_{14})x^2 + (a_9 + a_{13} + a_{15})x+ \\ &\quad (a_8 + a_{12} + a_{14} + a_{15}); \\ r(x) &= (a_7 + a_{11} + a_{13} + a_{14} + a_{15})x^7 + (a_6 + a_{10} + a_{12} + a_{13} + a_{14})x^6+ \\ &\quad (a_5 + a_9 + a_{11} + a_{12} + a_{13})x^5 + (a_4 + a_8 + a_{10} + a_{11} + a_{12})x^4+ \\ &\quad (a_3 + a_9 + a_{10} + a_{13} + a_{14} + a_{15})x^3 + (a_2 + a_8 + a_9 + a_{12} + a_{13} + a_{14})x^2+ \\ &\quad (a_1 + a_8 + a_{12} + a_{14} + a_{15})x + a_0. \end{aligned}$$

Let $a(x) = a_{15}x^{15} + a_{14}x^{14} + \ldots + a_1x + a_0$, then $a(x) = (x^8 + x^4 + x^2 + x + 1)q(x) + r(x)$ with

$$\begin{aligned} q(x) &= (a_8 + a_{12} + a_{14} + a_{15}) + (a_9 + a_{13} + a_{15})x+ \\ &\quad (a_{10} + a_{14})x^2 + (a_{11} + a_{15})x^3 + a_{12}x^4 + a_{13}x^5 + a_{14}x^6 + a_{15}x^7. \\ r(x) &= (a_7 + a_{11} + a_{13} + a_{14})x^7 + (a_6 + a_{10} + a_{12} + a_{13})x^6 + (a_5 + a_9 + a_{11} + a_{12})x^5+ \\ &\quad (a_4 + a_8 + a_{10} + a_{11})x^4 + (a_3 + a_9 + a_{10} + a_{11} + a_{13} + a_{14})x^3+ \\ &\quad (a_2 + a_8 + a_9 + a_{10} + a_{12} + a_{13})x^2 + (a_1 + a_8 + a_9 + a_{12} + a_{13} + a_{14})x+ \\ &\quad (a_0 + a_8 + a_{12} + a_{14} + a_{15}) \end{aligned}$$

A.3 Calculation modulo degree 4 polynomials

Let $a(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$, then $a(x) = (x_4 + x + \lambda)q(x) + r(x)$ with

$$\begin{aligned} &q(x) = a_7x^3 + a_6x^2 + a_5x + a_7 + a_4; \\ &r(x) = (a_3 + a_6 + \lambda a_7)x^3 + (a_2 + a_5 + \lambda a_6)x^2 + (a_1 + a_4 + a_7 + \lambda a_5)x + a_0 + \lambda(a_4 + a_7). \end{aligned}$$

For $\lambda = 1$, then getting $r(x)$ costs 9 additions and we must add 4 multiplications if $\lambda \neq 1, 0$. If $\lambda = 0$, it costs 4 additions.

A.3.1 Calculation modulo degree 2 polynomials

Let $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$, then $a(x) = (x^2 + x + \lambda)q(x) + r(x)$ with

$$\begin{aligned} &q(x) = a_3x + a_2 + a_3; \\ &r(x) = (a_1 + a_2 + a_3 + \lambda a_3)x + a_0 + \lambda(a_2 + a_3). \end{aligned}$$

If $\lambda = 0$ or 1, then then getting $r(x)$ costs 2 additions and we must add 2 multiplications if $\lambda \neq 0, 1$.

A.3.2 Calculation modulo degree 1 polynomials

Let $a(x) = a_1x + a_0$, then $a(x) = (x + \lambda)q(x) + r(x)$ with

$$\begin{aligned} &q(x) = a_1; \\ &r(x) = a_0 + a_1\lambda. \end{aligned}$$

In average getting $r(x)$ costs 1 multiplication and 1 addition.

A.3.3 DFT cost for $d = 7$

Establishing the complexity, at this point depends of the considered material. For example, for hardware, it will be provided by the worst case at any stage, but we should not take in account the number of division since it is done in parallel.

Let A = Addition and B = Multiplication, then at the first stage, despite costs totally

$$S_1 = 4A + 9A + 2 \times (9A + 4B) = 31A + 8B.$$

Expansive division costs $9A + 4B$.

The second stage cost totally

$$S_2 = 2A + 2A + 6 \times (2A + 2B) = 16A + 12B.$$

Expansive division costs $2A + 2B$. The last stage costs totally

$$S_3 = A + 13 \times (A + B) = 14A + 13B.$$

Expansive division costs $A + B$.

Finally the cost of this DFT is $61A + 33B$.

On the hardware complexity point of view the cost is $12A + 7B$.

The theoretical complexity predicted $15 \log(15) \approx 40.6$ multiplications and $\frac{15}{4} \log^2(15) \approx 20.7$ additions.

A.4 Building DFT between $n = 5$ and $n = 15$

In fact our DFT of a polynomial P consists in evaluating it over a set of points $\alpha_i \in \mathbb{F}_{2^8},\ i \in \{1, \ldots, n\}$. The inverse of the DFT is given by the evaluation of P over the set $\alpha_i^{-1} \in \mathbb{F}_{2^8},\ i \in \{1, \ldots, n\}$.

References

[BBD$^+$15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. 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 457–485. Springer, 2015.

[BGK04] Johannes Blömer, Jorge Guajardo, and Volker Krummel. Provably secure masking of AES. In Helena Handschuh and M. Anwar Hasan, editors, *Selected Areas in Cryptography, 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers*, volume 3357 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2004.

[BHP98] Richard E Blahut, WC Huffman, and V Pless. Decoding of cyclic codes and codes on curves. *Handbook of coding theory*, 2:1569–1633, 1998.

[Can89] David G Cantor. On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A*, 50(2):285–300, 1989.

[CCG$^+$20] Claude Carlet, Wei Cheng, Kouassi Goli, Jean-Luc Danger, and Sylvain Guilley. Detecting Faults in Inner Product Masking Scheme IPM-FD: IPM with Fault Detection (Extended version). *Journal of Cryptographic Engineering*, page 15, May 30 2020. DOI: 10.1007/s13389-020-00227-6.

[CPRR15] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 742–763. Springer, 2015.

[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.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 445–465. Springer, 2010.

[Gao03] Shuhong Gao. A new algorithm for decoding reed-solomon codes. In *Communications, information and network security*, pages 55–68. Springer, 2003.

[GJR18] Dahmun Goudarzi, Antoine Joux, and Matthieu Rivain. How to Securely Compute with Noisy Leakage in Quasilinear Complexity. 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 547–574. Springer, 2018.

[GM10] Shuhong Gao and Todd D. Mateer. Additive Fast Fourier Transforms Over Finite Fields. *IEEE Trans. Inf. Theory*, 56(12):6265–6272, 2010.

[GPRV21] Dahmun Goudarzi, Thomas Prest, Matthieu Rivain, and Damien Vergnaud. Probing security through input-output separation and revisited quasilinear masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):599–640, 2021.

[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.

[Jr.65] G. David Forney Jr. On decoding BCH codes. *IEEE Trans. Inf. Theory*, 11(4):549–557, 1965.

[JT12] Marc Joye and Michael Tunstall, editors. *Fault Analysis in Cryptography.* Information Security and Cryptography. Springer, 2012.

[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.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. 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 388–397. Springer, 1999.

[Knu11] Donald E. Knuth. *The Art of Computer Programming.* Addison Wesley, March 2011. ISBN-13: 978-0201038040.

[LCK$^+$18] Wen-Ding Li, Ming-Shing Chen, Po-Chun Kuo, Chen-Mou Cheng, and Bo-Yin Yang. Frobenius Additive Fast Fourier Transform. In Manuel Kauers, Alexey Ovchinnikov, and Éric Schost, editors, *Proceedings of the 2018 ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2018, New York, NY, USA, July 16-19, 2018*, pages 263–270. ACM, 2018.

[Mas69] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory*, 15(1):122–127, 1969.

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

[MS77] Florence Jessie MacWilliams and N. J. A. Neil James Alexander Sloane. *The theory of error correcting codes.* North-Holland mathematical library. North-Holland Pub. Co. New York, Amsterdam, New York, 1977. Includes index.

[Pet60] W. Wesley Peterson. Encoding and error-correction procedures for the bose-chaudhuri codes. *IRE Trans. Inf. Theory*, 6(4):459–470, 1960.

[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.

[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.

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

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

[TL20] Nianqi Tang and Yun Lin. Fast Encoding and Decoding Algorithms for Arbitrary (n, k) Reed-Solomon Codes Over $\mathbb{F}_{2^m}$. *IEEE Commun. Lett.*, 24(4):716–719, 2020.

[Uni] University of Sydney (Australia). Magma Computational Algebra System. http://magma.maths.usyd.edu.au/magma/, Accessed on 2022-08-22.

[vzGG96] Joachim von zur Gathen and Jürgen Gerhard. Arithmetic and Factorization of Polynomial over $\mathbb{F}_2$ (Extended Abstract). In *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*, ISSAC '96, page 1–9, New York, NY, USA, 1996. Association for Computing Machinery.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra (3. ed.)*. Cambridge University Press, 2013.

[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.

[WZ88] Yao Wang and Xuelong Zhu. A fast algorithm for the Fourier transform over finite fields and its VLSI implementation. *IEEE J. Sel. Areas Commun.*, 6(3):572–577, 1988.

[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.