IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 1, pp. 398–432. DOI:10.46586/tches.v2024.i1.398-432

Quasi-linear masking against SCA and FIA, with cost amortization

Claude Carlet[1,2] Abderrahman Daif[3], Sylvain Guilley[4,5] and Cédric Tavernier[6]

[1] University of Bergen, Bergen, Norway,
[2] LAGA, Department of Mathematics, University of Paris 8 (and Paris 13 and CNRS), Saint–Denis Cedex 02, France,
[3] BULL SAS, Les Clayes-sous-Bois, France,
[4] Secure-IC S.A.S., Paris, France,
[5] Telecom Paris, Institut Polytechnique de Paris, Palaiseau, France,
[6] Hensoldt France, Plaisir, France,

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. Against side-channel attacks, they are characterized by SNI 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. The security paradigm is that of code-based masking. Coding theory is amenable both to mask material at a prescribed order, by mixing the information, 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. We provide a security proof, leveraging both coding and probing security arguments. Regarding fault detection, our masking is capable of detecting up to d faults, where $2d + 1$ is the length of the code, at any place of the algorithm, including within gadgets. In addition to the theory, that makes use of the Frobenius Additive Fast Fourier Transform, we show performance results, in a C language implementation, which confirms in practice that the complexity is quasi-linear in the code length.

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

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.

Received: 2023-07-15 Accepted: 2023-09-15 Published: 2023-12-04

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 linear 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 before 2020, it seemed difficult to mask one element of the field $\mathbb{F}_{q^\ell}$ in a way ensuring a d^{th}-order probing security, with a better complexity than $\mathcal{O}(d^2)$ multiplications over $\mathbb{F}_{q^\ell}$. Recently, leveraging Karatsuba multiplication, Maxime Plançon [Pla22] introduced RTIK masking scheme. This masking style manages to get reduced complexity down to $\mathcal{O}(d^{\log_2(3)})$, i.e., $\mathcal{O}(d^{1.59})$, for limited values of d only (namely d being an extension order of the field where computations takes place, when this field happens to be an extension).

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]). Quasi-linear masking enables significant performance improvements on masking schemes which considerably ease their adoption by the industry. Unfortunately the NTT works only for prime fields with odd characteristics and large orders 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]. (Notice that this DFT is "general" in that it operates on finite fields.) 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 (in the field) are no longer transparent. Instead of having a complexity $\mathcal{O}(n)$ (linear in the number n of shares), because each share is applied the linear transformation on itself, individually, an operation of quasi-linear complexity 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 (vector space) 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 in the same article relevant to describe the capability to detect faults on top of a masking scheme: indeed, when the two vector spaces 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. Notice that CBM class includes as special cases Boolean masking and inner product masking.

1.2 Analysis of the state of the art

We begin in this subsection with a comparison with state-of-the-art of combined side-channel and fault injection attacks. The efficiency of side-channel analysis is captured by the masking complexity and the ability to mask several symbols at the same time (denoted "cost am." for "cost amortization"). Only our proposal enjoys this cost amortization capability. The efficiency of the protection against fault injection is qualified according to:

1. whether the detection is end-to-end throughout the algorithm;
2. whether the detection needs to be performed at pre-defined checkpoints set at design time or whether no detection is required (e.g., when faults are infective thereby preventing an attack to exploit them). Notice that checkpoints may be placed at strategic waypoints during the execution of the algorithm, or only at the end prior to disclosing the demasked result.

Most known masking countermeasures apply either to binary fields or to prime fields, whereas our masking can handle both binary and prime fields (and actually any finite field in general).

Table 1: Comparison of our masking scheme with the state of the art

Scheme name		Side-channel protection		Fault protection		Field
		Complexity	Cost am.	End-to-end	Detection	
ParTI	[SMG16]	Quadratic $(\mathcal{O}(d^2))$	No	Yes	At checkpoints	$\mathbb{F}_2$
CAPA	[RMB+18]	Quadratic $(\mathcal{O}(d^2))$	No	Yes	At checkpoints	$\mathbb{F}_2$
GJR	[GJR18]	Quasi-linear $(\mathcal{O}(d \log d))$	No	No	N/A	$\mathbb{F}_p$
M&M	[MAN+19]	Quadratic $(\mathcal{O}(d^2))$	No	Yes	Infective	$\mathbb{F}_2$
DOMREP	[GPK+21]	Quadratic $(\mathcal{O}(d^2))$	No	Yes	At checkpoints	$\mathbb{F}_2$
GJR+	[GPRV21]	Quasi-linear $(\mathcal{O}(d \log d))$	No	No	N/A	$\mathbb{F}_q$
CINI MINIS	[FRSG22]	Quadratic $(\mathcal{O}(d^2))$	No	Yes	At checkpoints	$\mathbb{F}_2$
RTIK	[Pla22]	Polynomial $(\mathcal{O}(d^{\log_2 3}))$	No	No	N/A	$\mathbb{F}_2$
SotA / laOla	[BEF+23]	Quadratic $(\mathcal{O}(d^2))$	No	Yes	At checkpoints	$\mathbb{F}_q$
Our work		Quasi-linear $(\mathcal{O}(d \log d))$	Yes	Yes	At checkpoints	$\mathbb{F}_q$

The comparison is given in Tab. 1. Regarding the applicable field, the different fields are denoted by $\mathbb{F}_2$ vs $\mathbb{F}_q$, where q stands for any prime power. Masking schemes compatible with $\mathbb{F}_q$ are thus more versatile.

We analyze now 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, none of these papers addresses 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". These masking orders are non-practical. Indeed, in real life, masking order is rather low, such as 1, 2 or maximum 3.

Complex implementation. The technique of [GPRV21] involves a randomized Fourier transform. Namely, the primitive root of unit which defines the Fourier transform must be chosen at random (see page 602). This is an obvious limitation in terms of efficiency: the DFT operations must be pre-computed prior to any cryptographic masked operation (whereas our scheme does not require any pre-computation).

Abstract specification. In [GPRV21], the DFT is not instantiated, which limits the ability to compare with other schemes, apple to apple, in terms of actual performances (actually [GPRV21] only provides data complexities). 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, and fault detection/correction.

Proofs of security against SCA and FIA based on code 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 show 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 ($t \geq 1$, and $t > 1$ when cost amortization is enforced).

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 easily implementable. Namely, we provide performance characterization in C language. In particular, it supports the effectiveness of cost amortization. Our benchmarks are on the block cipher AES, but our masking can apply as well to lattice-based post-quantum cryptographic algorithms (such as Crystals Kyber and Dilithium, as explained in Sec. 8). We compare our performance results to others but rare are the papers on masking which actually indicated them with enough precision for allowing comparison.

1.4 Outline

Preliminary notions are given in Sec. 2. They focus on DFT computation as it is the most complex operation in our masking. We propose in Sec. 3 to consider an original DFT method proposed by Wang and Zhu 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 detect or correct errors and erasures of any codeword present anywhere in the process of the ciphering algorithm, including within gadgets. The security rationale is detailed in Sec. 6, where we provide formal proofs in the CBM and SNI models. Implementation in C language is given in Sec. 7, along with performance results. Some discussions are available in Sec. 8. Conclusions and perpectives are in Sec. 9.

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. $d = 7$). Those two values represent *regular* and *substantial/high* security levels.

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 be 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 (i.e. $\omega^n = 1$). 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 can be defined as that of the evaluation of the polynomial basis $1, X, X^2, \dots, X^{k-1}$ over the set $1, \omega, \omega^2, \dots, \omega^{n-1}$. 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 $(\omega^{ij})_{0\le i\le n-1, 0\le j\le n-1}$, known as the Vandermonde matrix defined over $1, \omega, \omega^2, \dots, \omega^{n-1}$, 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 $1, \omega^{n-1}, \omega^{n-2}, \dots, \omega^1$.

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 than or equal to d. The result is PQ, a polynomial of degree less than or equal to $2d$.

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

Every polynomial is evaluated over $\{1, \omega, \dots, \omega^{n-1}\}$. 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 $(p_0, p_1, \dots, p_{d-1})$ 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 $(P(1), P(\omega), \dots, P(\omega^{n-1}))$ with the inverse of the Vandermonde matrix.

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

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

Then the DFT inverse is defined by:

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

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].

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 implementations. 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^ℓ where ℓ is a power of p. This method is generalized by von zur Gathen and Gerhard [vzGG96] to arbitrary ℓ. 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 $n=2^\ell$, Gao and Mateer [GM10] recently improved on Cantor's method. When ℓ is a power of 2, the above time complexity can be improved to $\mathcal{O}(n\log(n)\log\log(n))$. For arbitrary ℓ, 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 $q_{0,0} = X$. 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:

$$\begin{tikzcd}
 & & & q_{2,0} = X^4 + X \arrow[dll] \arrow[drr] & & & \\
 & q_{1,0} = X^2 + X \arrow[dl] \arrow[dr] & & & & q_{1,1} = X^2 + X + 1 \arrow[dl] \arrow[dr] & \\
q_{0,0} = X & & q_{0,1} = X + 1 & & q_{0,2} = X + \omega & & q_{0,3} = X + \omega + 1
\end{tikzcd}$$

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: $a = (a_0, a_1, \ldots, a_{n-1})$
  Output: $(b_0, b_1, \ldots, b_{n-1})$ the DFT of $a$
1 $P_{\lceil \log_2(n) \rceil, 0} \leftarrow \sum_{i=0}^{n-1} a_i X^i$
2 for $i \in \{\lceil \log_2(n) \rceil - 1, \lceil \log_2(n) \rceil - 2, \ldots, 0\}$ do
3     for $j \in \{1, \ldots, 2^{\lceil \log_2(n) \rceil - i}\}$ do
4         $P_{i,j} \leftarrow P_{i+1, \lfloor j/2 \rfloor} \bmod q_{i,j}$
5 return $(P_{0,j})_{0 \le j \le n-1}) = (b_0, b_1, \ldots, b_{n-1})$
```

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 $r_0, \ldots, r_{d-1}$ in $\mathbb{F}_q$ and encode the vector $\vec{a} = (x, r_0, \ldots, r_{d-1}, 0, \ldots, 0) \in \mathbb{F}_q^n$ with the Vandermonde matrix. We define:

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

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

- $\mathtt{mask}(x + x') = \vec{z} + \vec{z}\,'$,
- $\mathtt{mask}(\lambda x) = \lambda \cdot \vec{z}$ 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 $\mathtt{mask}(xx')$ without unmasking x or x'. We denote $\vec{y} = \vec{z} * \vec{z}\,' := (z_j z'_j)_{j\in\{0,\dots,2d\}}$ where "$*$" is the term-to-term product between two vectors. For $j \in \{1, \dots, 2d\}$, we have:

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

The coefficients r''_i are obtained from the multiplication between $Z(X) = x + \sum_{i=1}^{d} r_i X^i$ and $Z'(X) = x' + \sum_{i=1}^{d} r'_i X^i$. Namely,

$$r''_i = \begin{cases} \sum_{1\le k,l\le d,\ \text{s.t. } k+l=i} r_k r'_l + x r'_i + x' r_i & \text{when } 1 \le i \le d, \\ \sum_{1\le k,l\le d,\ \text{s.t. } k+l=i} r_k r'_l & \text{when } d+1 \le i \le 2d. \end{cases}$$

The multiplication between $Z(X)$ and $Z'(X)$ of degree d gives a polynomial $Y(X) = xx' + \sum_{i=1}^{2d} r''_i X^i$ of degree $2d$. Thus, to get $\mathtt{mask}(xx')$ we need to eliminate the coefficients r''_i for $i \in \{d+1, \dots, 2d\}$.

3.3.1 Extracting the last coefficients

We have:

$$\begin{array}{rcl} Y(X) &=& xx' + \sum_{i=1}^{2d} r''_i X^i = xx' + \sum_{i=1}^{d} r''_i X^i + \sum_{i=d+1}^{2d} r''_i X^i \ . \\ \Longrightarrow \vec{y} &=& \mathrm{DFT}(xx', r''_1, \dots, r''_d, 0, \dots, 0) + \mathrm{DFT}(0, \dots, 0, r''_{d+1}, \dots, r''_{2d}) \ . \\ &=& \mathtt{mask}(xx') + DFT(0, \dots, 0, r''_{d+1}, \dots, r''_{2d}) \ . \\ \Longrightarrow \mathtt{mask}(xx') &=& \vec{y} + DFT(0, \dots, 0, r''_{d+1}, \dots, r''_{2d}) \ . \end{array}$$

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

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

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

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

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

$$\begin{array}{rcl} r''_{j+d+1} &=& \sum_{i=0}^{2d} y_i \omega^{-i(j+d+1)} \\ r''_{j+d+1} &=& \sum_{i=0}^{2d} y_i \omega^{-i(d+1)} \omega^{-ij} \\ \Longrightarrow \vec{r}\,'' &=& \mathrm{IDFT}(\vec{w}) \end{array}$$

where $\vec{w} = (y_i \omega^{-i(d+1)})_{0\le i\le 2d}$.

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

```
   Input: a vector $\vec{y} \in \mathbb{F}_q^n$
   Output: $\vec{r}\,'' \in \mathbb{F}_q^n$
1  Build the vector $\vec{w} = (y_i \omega^{-i(d+1)})_{0 \le i \le 2d}$
2  return $\vec{r}\,'' = IDFT(\vec{w})$
```

Table 2: Complexity of operations involved in the masked multiplication

Variable	Cost
$\vec{y}$	n
$\vec{r}\,''$	$n + n\log(n)$
$\mathtt{mask}(xx')$	$2n(1 + \log(n))$

3.3.2 Algorithm for the masked multiplication

We get:

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

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. 2.

Algorithm 3: `oneElementMultiplication` Complexity: $n(d + 1 + \log(n))$

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

In conclusion, the complexity of addition is linear, that of multiplication is quasi-linear. Besides, masking and demasking each 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.

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 set $\{u_0, u_1, \dots, u_{2d}\} \in \mathbb{F}_q^{d+1}$ with $u_i \neq u_j$ $\forall i, j \in \{0, \dots, 2d\}$ and such that

$$\{u_0, u_1, \dots, u_{2d}\} \cap \{1, \omega, \omega^2, \dots, \omega^{n-1}\} = \varnothing. \tag{1}$$

We want now to mask the vector $\vec{x} = (x_0, \dots, x_{t-1}) \in \mathbb{F}_q^t$ with $1 \le t < d$. (the case $t = 1$ has been addressed in previous section 3.)

4.1 Encoding procedure

First we pick randomly $\vec{r} = (r_{t+1}, r_{t+2}, \dots, r_{d+1})$ in $\mathbb{F}_q^{d+1-t}$. By Lagrange interpolation, there exists a vector $\vec{a} = (a_0, a_1, \dots, a_d)$ and the associated polynomial $P_{\vec{x}}(X) = a_0 + a_1X + \cdots + a_dX^d$ of degree at most d that satisfies $P_{\vec{x}}(u_i) = x_i$ for $i \in \{0, \dots, t-1\}$ and $P_{\vec{x}}(u_i) = r_i$ for $i \in \{t, \dots, d\}$.

Let us define the matrix $A \in \mathbb{F}_q^{(d+1)\times(d+1)}$, where $A_{i,j} = u_j^i$ for any $i, j \in \{0, \dots, d\}$. This matrix is a Vandermonde matrix which is invertible since $u_i \neq u_j$ for $i \neq j$. Then we have:

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

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

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

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 the 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 $P''(X) = a_tX^t + \cdots + a_dX^d$, then we evaluate $P'(X) = a_0 + a_1X + \cdots + a_{t-1}X^{t-1}$ over $u_0, u_1, \dots, u_{t-1}$ which costs $t(d-t)$ multiplications over $\mathbb{F}_q$.

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

$$\begin{aligned}
\underbrace{\begin{bmatrix} a_0 & \dots & a_{t-1} \end{bmatrix}}_{\vec{a}\,'} \times A' &= \begin{bmatrix} x_0 + P''(u_0) & \dots & x_i + P''(u_i) & \dots & x_{t-1} + P''(u_{t-1}) \end{bmatrix} \\
&= \vec{x} + \begin{bmatrix} P''(u_0) & \dots & P''(u_i) & \dots & P''(u_{t-1}) \end{bmatrix} \\
&= \vec{x} + \underbrace{\begin{bmatrix} a_t & \dots & a_d \end{bmatrix}}_{\vec{a}\,''} \times A'' = \vec{x} + \vec{a}\,'' \times A'' ,
\end{aligned}$$

where:

- $A' \in \mathbb{F}_q^{t\times t}$, and $A'_{i,j} = A_{i,j}$ for any $0 \le i, j < t$;
- $A'' \in \mathbb{F}_q^{(d+1-t)\times t}$, $A''_{i,j} = A_{i+t,j}$ for any $0 \le i < d+1-t$ and $0 \le j < t$;
- $\vec{a}\,'' \in \mathbb{F}_q^{d+1-t}$ is a random vector.

Thus, the calculation of $\vec{a} = (\vec{a}\,' \mid \vec{a}\,'') = \left((\vec{x} + \vec{a}\,'' \times A'') \times A'^{-1} \mid \vec{a}\,''\right)$ 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 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($\vec{x}$) $\in \mathbb{F}_q^n$
1 $\vec{a}\,'' = (a_t, a_{t+1}, \dots, a_d) \xleftarrow{\$} \mathbb{F}_q^{d+1-t}$
2 $\vec{a}\,' \leftarrow (\vec{x} + \vec{a}\,'' \times A'') \times A'^{-1}$
3 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 $\mathtt{mask}(\vec{0})$, namely

$$\mathtt{refresh}(\mathtt{mask}(\vec{x})) = \mathtt{mask}(\vec{x}) + \mathtt{mask}(\vec{0}). \tag{2}$$

4.2 Masking the multiplication

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

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

where the '$*$' operation stands for the pairwise product.

The polynomial obtained by performing $\mathrm{DFT}_\omega^{-1}(\mathrm{DFT}_\omega(P_{\vec{x}}) \times \mathrm{DFT}_\omega(P_{\vec{x}'})) = P_{\vec{x}}(X) \times P_{\vec{x}'}(X) = C(X) = \sum_{i=0}^{2d} c_i X^i$ is a $2d$ degree polynomial, which satisfies $C(u_i) = P_{\vec{x}}(u_i) \times P_{\vec{x}'}(u_i) = x_i x'_i$ 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: $D(u_i) = C(u_i)$ for all $0 \leq i \leq t-1$.

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

$$\begin{aligned} D(X) &= c_0 + c_1 X + \ldots + c_d X^d + u_0^d(c_{d+1}X + \ldots + c_{2d}X^d) \\ &= c_0 + (c_1 + \alpha^d c_{d+1})X + \cdots + (c_d + \alpha^d c_{2d})X^d\ . \end{aligned}$$

Obviously, in this case $D(u_0) = C(u_0) = x_1 x'_1$. We propose to generalize this construction. Let:

$$U_j(X) = u_j^d \frac{(X-u_0)\cdots(X-u_{j-1})(X-u_{j+1})\cdots(X-u_{t-1})}{(u_j-u_0)\cdots(u_j-u_{j-1})(u_j-u_{j+1})\cdots(u_j-u_{t-1})}.$$

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

$$\begin{aligned} D(X) = c_0 + c_1 X + \cdots + c_d X^d \quad &+ \quad \textstyle\sum_{j=0}^{t-1} U_j(X)(c_{d+1}X + \cdots + c_{2d-t+1}X^{d-t+1}) \\ &+ \quad \textstyle\sum_{j=0}^{t-1} U_j(X) \sum_{i=1}^{t-1} c_{2d-t+1+i} u_j^{d-t+1+i}. \end{aligned}$$

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

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

$$\begin{aligned} D(X) \quad = c_0 + c_1 X + \cdots + c_d X^d \quad &+ \quad (c_{d+1}X + \cdots + c_{2d-t+1}X^{d-t+1}) \textstyle\sum_{j=0}^{t-1} U_j(X) \\ &+ \quad \textstyle\sum_{i=1}^{t-1} c_{2d-t+1+i} \sum_{j=0}^{t-1} U_j(X) u_j^{d-t+1+i}. \end{aligned}$$

Thus:

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

where $G_i = \mathrm{DFT}_\omega(\sum_{j=0}^{t-1} U_j(X) u_j^{d-t+1+i})$ for $i \in \{1, \ldots, t-1\}$ and $\vec{U} = \mathrm{DFT}_\omega(\sum_{j=1}^{t} U_j(X))$ are pre-computed values, and we define now how to build the last coefficients $c_{d+1}, \ldots, c_{2d}$ without revealing some sensitive information. If we denote $\vec{y} = (C(\omega^i))_{i \in [\![0..2d]\!]}$, then we have $\mathrm{IDFT}_\omega(y) = (c_0, \ldots, c_d, \ldots, c_{2d})$ and by definition, for $0 \leq j \leq d-1$,

$$\begin{aligned} c_{j+d+1} \quad &= \quad \textstyle\sum_{i=0}^{2d} y_i \omega^{-i(j+d+1)} \\ &= \quad \textstyle\sum_{i=0}^{2d} \left(y_i \omega^{-i(d+1)}\right) \omega^{-ij} \end{aligned}$$

where $\vec{w} = (y_i \omega^{-i(d+1)})_{0 \le i \le 2d}$. Then we can calculate:

$$\vec{c} = \Big(c_{d+1}, \dots, c_{2d}, \dots\Big) = \Big(\mathrm{IDFT}(\vec{w})\Big). \tag{3}$$

This computation is formalized as a routine in Alg. 2, which indeed extracts the coefficients of largest degree (from $d+1$ to $2d$).

If we denote

$$\begin{array}{rcl} \phi(C,\omega) & = & \mathrm{DFT}_\omega(c_{d+1}X^{d+1} + \cdots + c_{2d}X^{2d}) \\ & & +\mathrm{DFT}_\omega(c_{d+1}X + \cdots + c_{2d-t+1}X^{d-t+1}) * \vec{u} \;\;, \\ & & + \sum_{i=1}^{t-1} c_{2d-t+1+i} \cdot G_i \end{array}$$

we get

$$\mathtt{mask}(\vec{x} * \vec{x}') = \mathtt{mask}(\vec{x}) * \mathtt{mask}(\vec{x}') + \phi(C,\omega).$$

Algorithm 5: `severalByteProduct` Complexity: $n(3 + t + 4\log(n))$

```
Input: two vectors $\vec{z} = \mathtt{mask}(\vec{x}) \in \mathbb{F}_q^n$ and $\vec{z}\,' = \mathtt{mask}(\vec{x}\,') \in \mathbb{F}_q^n$
Output: $\mathtt{mask}(\vec{x} * \vec{x}\,') \in \mathbb{F}_q^n$
1  $\vec{y} \in \mathbb{F}_q^n$
2  for $i \in \{0, \dots, n-1\}$ do
3      $y_i \leftarrow z_i z'_i$
4  $\vec{c}\,'' = \mathtt{ExtractLastCoefficients}(\vec{y}) = (c_{d+1}, \dots, c_{2d})$
5  $\vec{c} \leftarrow (0, \dots, 0 \mid \vec{c}\,'') = (0, \dots, 0, c_{d+1}, \dots, c_{2d}) \in \mathbb{F}_q^n$.
6  $\vec{v} \leftarrow \vec{0} \in \mathbb{F}_q^n$
7  for $0 \le i < t-1$ do
8      for $0 \le j < n$ do
9          $v_j \leftarrow v_j + G_{i+1,j} \cdot c_{2d-t+2+i}$
10 $\vec{c}\,' \leftarrow \vec{0} \in \mathbb{F}_q^n$
11 for $i \in \{1, \dots, d-t+1\}$ do
12     $c'_i \leftarrow c_{d+i}$
13 $\vec{w}\,' \leftarrow \mathrm{DFT}(\vec{c}\,') \in \mathbb{F}_q^n$
14 for $i \in \{0, \dots, n-1\}$ do
15     $w'_i \leftarrow w_i u_i$
16 return $\mathtt{refresh}(\vec{y} + \mathtt{DFT}(\vec{c}) + \vec{w}\,' + \vec{v})$
```

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 $L \in K^{t \times t}$ a public matrix, we need to construct an algorithm `MatrixProduct` such that:

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

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

$$\mathtt{mask}(\vec{x}) = (\vec{x}, \vec{r}, \vec{0}) \cdot N \;. \tag{4}$$

where:

$$N = \begin{bmatrix} A^{-1} & 0 \\ 0 & 0 \end{bmatrix} \times M \in \mathbb{F}_q^{n\times n} \ .$$

Let us denote $L' = N^{-1} \cdot \begin{bmatrix} L & 0 \\ 0 & I \end{bmatrix} \cdot N$, we have:

$$\begin{aligned} \texttt{mask}(\vec{x}) \cdot L' &= (\vec{x}, \vec{r}, \vec{0}) \cdot (N \cdot L') \\ &= (\vec{x}, \vec{r}, \vec{0}) \cdot (N \cdot N^{-1} \cdot \begin{bmatrix} L & 0 \\ 0 & I \end{bmatrix} \cdot N) \\ &= (\vec{x} \cdot L, \vec{r}, \vec{0}) \cdot N \\ &= \texttt{mask}(\vec{x} \cdot L) \ . \end{aligned}$$

Thus:

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

4.4 Exponentiation algorithm

Let e be a power of 2, we denote $\vec{x}^{\ e} = (x_1^e, \ldots, x_{t+1}^e) \in \mathbb{F}_q^{t+1}$. In order to calculate `SubBytes` transformation efficiently we need to calculate $\texttt{mask}(\vec{x}^{\ e})$ (see for instance [RP10, Alg. 3]). We have:

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

In this case, the order of the operations is very important. As a matter of fact, the $\texttt{mask}(\vec{x})^e \cdot (N^e)^{-1}$ can divulge the sensitive data if it has been done as indicated above. This is why it is mandatory to pre-compute $((N^e)^{-1} \times N)$ first (once for all), and only 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:

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

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 a Reed-Solomon code of parameters $[n = 2d+1, k = d+1, d+1]_q$. 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 $e = (e_0, e_1, \ldots, e_{n-1}) = \mathrm{DFT}_\omega^{-1}(e(X))$ where $e(X)$ is a degree $n-1$ polynomial over $\mathbb{F}_q$. We denote by ϵ the weight of the non-zero coefficients (positions) in $e(X)$. 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 $\lambda = (\lambda_j)_{j\in[\![0,n-k-1]\!]}$ 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

$$E(X)\Lambda(X) = 0 \bmod X^n - 1. \tag{5}$$

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 loosing in generality, $\Lambda(X)$ can be normalized by setting $\lambda_0 = 1$. Equation (5) 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:

$$\underbrace{\begin{bmatrix} S_0 & S_1 & \ldots & S_{\epsilon-1} \\ & & \vdots & \\ S_i & S_{i+1} & \ldots & S_{\epsilon+i-1} \\ & & \vdots & \\ S_{n-k-er-1} & S_{n-k-\epsilon} & \ldots & S_{n-k-2} \end{bmatrix}}_{S} \times \underbrace{\begin{bmatrix} \Lambda_\epsilon \\ \vdots \\ \Lambda_i \\ \vdots \\ \Lambda_1 \end{bmatrix}}_{\Lambda} = \underbrace{\begin{bmatrix} -S_\epsilon \\ \vdots \\ -S_i \\ \vdots \\ -S_{n-k-1} \end{bmatrix}}_{T}. \tag{6}$$

Table 3: Side-channel security order *versus* fault detection / correction, in $\mathbb{F}_{256}$

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 ⋮ 7	7 6 ⋮ 1	7	3
17	8	1 2 ⋮ 8	8 7 ⋮ 1	8	3

Obviously, a unique solution 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 (6), 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 $X_j = \omega^{i_j}$:

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

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 3.

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:

$$\texttt{BLLC}(\mathbb{F}_q, 2\times k, d) \leq 2 \times \texttt{BLLC}(\mathbb{F}_q, k, d). \tag{7}$$

For instance, on $\mathbb{F}_{256}$, RS codes are minimum distance separable (MDS) and thus $\texttt{BLLC}(\mathbb{F}_q, k, d) = k + d - 1$. Thus Eqn. (7) 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 multiplication gadget and our capacity to detect fault injections during the computation steps of the encryption algorithm.

6.1 The encoding procedure

We remind that our encoding procedure of a vector $\vec{x} = (x_0, \ldots, x_{t-1})$ has been defined in subsection 4.1. It consists in picking randomly $\vec{r} = (r_{t+1}, r_{t+2}, \ldots, r_{d+1})$ in $\mathbb{F}_q^{d+1-t}$ and performing the operation:

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

We also recall that the matrix $A = (u_i^j)_{i,j \in [\![0..d]\!]}$. A first approach consists in showing 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 a multiplication by one Vandermonde matrix. This matrix happens to be the generator matrix of the Reed-Solomon code RS$[n, n, 1]$ defined over $\mathbb{F}_q$. A generator matrix of this code is defined by the evaluation of the monomials $(X^i)_{i \in \{0, n-1\}}$ over $1, \omega, \ldots, \omega^{n-1}$. 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,

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

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 sensitive information.

We can rewrite our encoding procedure as follows:

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

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

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

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

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

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

If we denote by d' the minimal distance of $\mathcal{C}_{H^{perp}}$: $d' = d_{min}(\mathcal{C}_{H^{perp}})$, then, as explained in [WMCS20], a direct consequence is that the encoding procedure enc_B is d'-private. Our task consists now in evaluating d' and we propose to demonstrate the following theorem:

Theorem 1. *Let an integer t, $1 \leq t \leq d$, a Vandermonde matrix A of the form $(u_j^i)_{i,j \in [\![0,d]\!]}$ with $u_i \neq u_j$. Let R the generator matrix of the Reed-Solomon code $RS[2d+1, d+1, n-d]$ of the form $(\omega^i j)_{i \in [\![0,d]\!], j \in [\![0,2d]\!]}$. We denote*

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

Let $\mathcal{C}_H$ the code generated by H, then, $d_{min}\left(\mathcal{C}_H^{\perp}\right)$ the minimal distance of $\mathcal{C}_H^{\perp}$ satisfies

$$d + 1 - t \leq d_{min}\left(\mathcal{H}^{\perp}\right) \leq d + 2 - t.$$

Proof. We denote by 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.

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 square matrix because it is equivalent (up to an 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 $A^{-1} \times A = Id_{d+1}$. By setting

$$A^{-1} = \begin{pmatrix} K'_{t\times(d+1)} \\ K_{(d+1-t)\times(d+1)} \end{pmatrix} \text{ and } A = \left(B_{(d+1)\times t}, B'_{(d+1)\times(d+1-t)}\right),$$

we get that

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

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

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

By construction ${}^t(B_{(d+1)\times t}) = {}^tB$ is a generator matrix of a code generated by the polynomials $1, X, X^2, \ldots, X^{t-1}$ defined over the set $u_0, \ldots, u_d$: this is a Reed-Solomon code $RS[d+1, t, d+2-t]$ of minimal distance $d+2-t$. We deduce that the encoder $(x, r) \mapsto (x, r)A^{-1}$ is a generic encoder of probing order $d+1-t$.

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 a 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 ${}^tB = (u_i^j)_{i\in[\![0..t-1]\!], j\in[\![0..d]\!]}$. Hence the code generated by D is an evaluation code generated by t independent 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 independent and we deduce that we have built the kernel of H. We have now to evaluate the minimal distance of this code $(T \cup D)$.

Hence, we have

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

Let

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

and

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

Then

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

For $i = 0$ (i.e $t = 1$), it means that the vector D_0 corresponds to the evaluation of the fraction

$$\frac{u_0^{d+1} X^{d+1} + X}{u_0 + X} \tag{8}$$

over $\{1, \omega, \ldots, \omega^{2d}\}$ and we are looking for a degree d polynomial $P(X)$ that cancels the maximum of positions of D_0, i.e. such that $Q(X) = (X + u_0)P(X) + X + u_0^{d+1} X^{d+1}$ 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 $2d + 1 - (d + 1) = d$. In the same time, the Singleton bound states that $d_{\min}(T \cup D_0) \leq 2d + 1 - (d + 1) + 1 = d + 1$. We deduce that for $D = D_0$,

$$d + 1 - t \leq d_{\min}(T \cup D) \leq d + 2 - t.$$

for $t = 2$, the Singleton bound states that $d_{\min}(T \cup D_0 \cup D_1) \leq 2d + 1 - (d + 2) + 1 = d = d + 2 - t$. We want to evaluate now the minimal distance of a codeword built from a linear combination of $D_{0,j}$, $D_{1,j}$ and T. It 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

$$P(X) = \frac{u_0^{d+1} X^{d+1} + X}{u_0 + X} + \theta \frac{u_1^{d+1} X^{d+1} + X}{u_1 + X}.$$

This is equivalent of studying the number of zero of the function $T(X) = (X + u_0)(X + u_1)P(X) + (X + u_1)(u_0^{d+1} X^{d+1} + X) + \theta(X + u_0)(u_1^{d+1} X^{d+1} + X)$. The degree of $T(X)$ is less or equal to $d + 2$ then $T(X)$ has $d + 2$ roots maximum which is equivalent to a minimal distance greater than $2d + 1 - (d + 2) = d - 1$ and we deduce:

$$d + 1 - t \leq d_{\min}(T \cup D) \leq d + 2 - t.$$

By induction we have that for any t, $d + 1 - t \leq d' \leq d + 2 - t$ and the probing security order is between $d - t$ and $d + 1 - t$, thus we have demonstrated Theorem 1. □

In this Theorem 1, we prove the security of the multiplicative gadget, including the transformation of the shares into the spectral domain (back and forth). This was left out of the scope of former work GJR+ [GPRV21]; we thus offer a comprehensive, end-to-end, security proof of the whole computation. Notice that in the section entitled "*Discussion on Hypothesis 1*", page 620 of [GPRV21], the announced security orders (obtained by exhaustive search, for some examplary small orders) are lower than our bound $d+2-t$. The reason is that examples in [GPRV21] do not satisfy condition (1).

Corollary 1. *Let $0 \leq i \leq d-1$. If u_i is such that $\frac{u_i^{d+1}X^{d+1}+X}{u_i+X}$ is a degree d polynomial, i.e $u_i + X$ divides $u_i^{d+1}X^{d+1}+X$, then we have $d' = d+2-t$.*

Proof. Without losing in generality, we can assume that $i = 0$. For $t = 1$ this is exactly the same proof than the previous one. For $t > 1$, we must evaluate the number of zeros of the function:

$$T(X) = \frac{u_0^{d+1}X^{d+1}+X}{u_0+X} + \theta_1 \frac{u_1^{d+1}X^{d+1}+X}{u_1+X} + \ldots + \theta_{t-1}\frac{u_{t-1}^{d+1}X^{d+1}+X}{u_{t-1}+X} .$$

As $\frac{u_0^{d+1}X^{d+1}+X}{u_0+X}$ is a degree d polynomial, then $(X+u_1)\cdots(X+u_{t-1})T(X)$ is a polynomial of degree less than $d+t-1$ which implies that the minimal distance is greater than $2d+1-d-t+1 = d+2-t$ and the singleton bound states that it is less than $d+2-t$ thus it is equal. □

This corollary shows that our masking scheme does reach the same masking order as [BEF+23].

Example 2. If $u_0 = 0$, we get $D_{0,j} = 1$ and the property is satisfied.

As summary, we have proven in this section that given a Vandermonde matrix $A = V(u) = (u_i^j)_{i,j\in[\![0..d]\!]}$, the encoder

$$x \mapsto (x, r) \mapsto (x, r)A^{-1}$$

is $d+1-t$ probing secured and if R is the generator matrix R of a Reed-Solomon code $RS[2d+1, d+1, d+1]$ with support in $\{1, \omega, \ldots, \omega^{n-1}\}$, then the composed encoder

$$x \mapsto (x, r) \mapsto (x, r)A^{-1}R$$

is at least $d-t$ probing secure.

6.2 NI and SNI criteria

Definition 1 ([MZ22])**.** A function f is t-NI if, when given a total of s outputs and internal probes, $s \leq t$ implies a dependency with maximum s input shares. A function f is t-SNI if $s \leq t$ implies a dependency with maximum i input shares, where i is the number of internal probes.

Corollary 2 ([WMCS20, Theorems 2 and 3])**.** *The scalar multiplication gadget is t-SNI and the addition gadget masking is t-NI.*

Proof. We remind that for complexity reason, we have replaced the classical Vandermonde matrix by the DFT algorithm. Our chosen DFT has a particular structure, it is an iterative DFT et each step corresponds to a matrix multiplication, then totally, our DFT corresponds to a classical encoder by a (sparse) matrix. Therefore, Theorems 2 and 3 of [WMCS20] apply verbatim. □

Remark 2. The refresh gadget of [GPRV21] is obviously compliant with our procedure and it is $(d-t)$-SNI.

To claim that the complete encoder with its associate gadgets is $(d-t)$-probing secured, we must prove the property for the multiplication gadget.

6.3 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. Regarding the security of this gadget. We remind that the authors of [GPRV21] made a strong hypothesis that we convert in a theorem:

Theorem 2 (Hypothesis (FFT Probing Security))**.** *The circuits processing*

$$DFT_\omega(x\|0) \mapsto r \text{ and } DFT_\omega^{-1}$$

are t_n^{DFT}-probing secure with $t_n^{DFT} \geq d-t$.

Proof. In fact the application $DFT_\omega(\vec{x}\|0) \mapsto r$ corresponds exactly to our masking operation $\texttt{mask}(\vec{x}) = (\vec{x}, \vec{r}) \times A^{-1} \times R$ except that A is more general than simply a Matrix of the form $(\alpha^{ij})_{i,j}$. We deduce that $t_n^{DFT} \geq d-t$ in this case since it corresponds to the theorem 1.
Regarding $\mathrm{DFT}_\omega^{-1} : u' \mapsto tt$: if fact, $u' = refresh(\texttt{mask}(\vec{x}) * \texttt{mask}(\vec{y}))$ where $*$ represents here the multiplication term by term and not the mask multiplication. In our masking, by definition, we have $u' = \texttt{mask}(\vec{0}) + \texttt{mask}(\vec{x}) * \texttt{mask}(\vec{y})$. $\texttt{mask}(\vec{0}) = \vec{r}H$ where $\vec{r}$ is a $d+1-t$ dimension vector which is random, then building $\vec{r}$ requires at least $d+1-t$ positions from the vector $\vec{r}H$. By construction, $\mathrm{DFT}_\omega^{-1}(\texttt{mask}(\vec{0})) = (a_0(r), a_1(r), \ldots, a_d(r), 0, \ldots, 0) = (0, r)A^{-1}$. Then $\mathrm{DFT}_\omega^{-1}(\texttt{mask}(\vec{x}) * \texttt{mask}(\vec{y})) = (c_0, \ldots, c_{2d})$. We deduce that:

$$tt = (c_0 + a_0(r), c_1 + a_1(r), \ldots, c_d + a_d(r), c_{d+1}, \ldots, c_{2d}).$$

We prove below this proof that we cannot construct a sensitive information from $(c_{d+1}, \ldots, c_{2d})$. The coefficients a_i of the vector $(c_0 + a_0(r), c_1 + a_1(r), \ldots, c_d + a_d(r))$ depends linearly of r. We have already proven that the encoder $(x, r)A^{-1}$ is $d+1-t$ probing secured, thus getting information from $(c_0 + a_0(r), c_1 + a_1(r), \ldots, c_d + a_d(r))$ requires to capture at least $d+1-t$ positions. We deduce the final result, the hypothesis is correct with $t_n^{DFT} = d-t$. □

Then, due to the the previous demonstrated hypothesis, we deduce the following lemma:

Lemma 1. *[GPRV21] The circuit processing* $(\texttt{mask}(x), \texttt{mask}(y)) \mapsto u = \texttt{mask}(x) * \texttt{mask}(y)$ *is $(d-t)$-probing secured.*

We provide here-after a proof by reduction of our Lemma 1 to the result formulated in [GPRV21, Lemma 1].

Proof. The authors of [GPRV21] have proven (page 619, lemma 1) that the following circuit processing

$$(x, y) \mapsto u = \mathrm{DFT}(x \parallel 0) * \mathrm{DFT}(y \parallel 0)$$

is t_n^{FFT} probing secure (In fact, we proved that the encoder $x \mapsto (x, r)A^{-1}$ is $d+1-t$ probing secure which implies that $t_n^{FFT} = d$ in [GPRV21] context) and we have proven that $\texttt{mask}(x) = \mathrm{DFT}((x, r)A^{-1} \parallel 0)$ is at least $d-t$ probing secure, then we can now apply the same proof, with $t_n^{FFT} = (d-t)$: either a probe gives some information about $\texttt{mask}(x)$ or about $\texttt{mask}(y)$. Finally, each position $M_t[i] = \texttt{mask}(x)[i] \times \texttt{mask}(y)[i]$ depends symmetrically of $\texttt{mask}(x)[i]$ and $\texttt{mask}(y)[i]$ which are independent and uniformly distributed, thus less than $d-t$ probes cannot give information about x and y. □

Remark 3 (Typographic mistake correction)*.* The proof of lemma 1 in [GPRV21] contains twice the argument "w is added to $\mathcal{W}_1$", whereas the second occurrence should read "w is added to $\mathcal{W}_2$".

We remind that the inner product $\texttt{mask}(x) * \texttt{mask}(y)$ here is not the gadget multiplication $\texttt{mask}(x) \times \texttt{mask}(y)$. Unfortunately, we cannot claim that $(\texttt{mask}(x), \texttt{mask}(y)) \mapsto \texttt{mask}(x) * \texttt{mask}(y)$ is $(d-t)$-NI or SNI secured because the function $(x, y) \mapsto x \cdot y$ does not satisfies the t-NI property and we cannot use the composition theorem.

The mask multiplication (gadget) is obtained from the following computation

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

where $G_i = \mathrm{DFT}_\omega(\sum_{j=0}^{t-1} U_j(X) u_j^{d-t+1+i})$ for $i \in \{1, \ldots, t-1\}$ and $\vec{U} = \sum_{j=1}^{t} U_j(X)$ are a pre-computed values. Then, it is clear that the computation of $\mathrm{DFT}_\omega(c_{d+1}X^{d+1} + \cdots + c_{2d}X^{2d})$, $\mathrm{DFT}_\omega\left((c_{d+1}X + \cdots + c_{2d-t+1}X^{d-t+1}) * \vec{U}\right)$ and $\sum_{i=1}^{t-1} c_{2d-t+1+i} \cdot G_i$ involves only the variables $c_{d+1}, \ldots, c_{2d-t+1}$ related to the sensitive information. Hence, the weakest side is obtained with the vector

$$(c_{d+1}, \ldots, c_{2d}) = \texttt{ExtractLastCoefficients}(\vec{z} * \vec{z}').$$

Then the question is: can we get information from $d-t$ position of the vector $(c_{d+1}, \ldots, c_{2d})$. Our claim is that our gadget is at least $d-t$ probing secured, then we must assume that in the model of attack, maximum $d-t$ values can be guessed from some measures. From $d-t$ pieces of knowledge from the vector $(c_{d+1}, \ldots, c_{2d})$, $x = \texttt{unmask}(z)$ and $x' = \texttt{unmask}(z')$ cannot be reconstructed: if an attacker has access to the following system of equations

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

We can evaluate the number of potential solutions for $(a_i)_{i \in [\![d..2d]\!]}$: by assuming 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 $k \leq d$. Thus totally this system admits at least $2^{m(d-1)(2^m-1)}$ solutions for d variables a_i. This result is obviously worst with less equations, thus this system of equation does not give information from $d+1-t$ values of (a_i) solutions.

We conclude that the gadget multiplication is $d-t$ probing secured.

Remark 4. It seems that our encoding method has similar properties than this one defined in [GPRV21] then it would be interesting to investigate if the region probing security still holds here.

6.4 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.

First of all, we cannot claim that our method is fully resilient against fault attack because we did not study the impact of generating a fault on the checker itself (the syndrome calculation), however, we show in this paper that we harden considerably the resilience against fault injection.

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.

Furthermore, 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, especially the Red-Solomon code involved in the gadget multiplication. 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 the scope of this paper.

We recall that by construction, 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. It can then give information 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, hence we propose to show in the next subsections that our error detection based on the syndrome decoding is secured and efficient.

We recall that we have:

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

where $(c_d, \ldots, c_{2d}) = \mathtt{ExtractLastCoefficients}(\mathtt{mask}(\vec{x}) * \mathtt{mask}(\vec{x}\,'))$, with $\vec{U}_k$ and G_i that are precomputed and we have denoted

$$\mathtt{mask}(\vec{x} * \vec{x}\,') = \mathtt{mask}(\vec{x}) * \mathtt{mask}(\vec{x}\,') + \phi(C, \omega).$$

Obviously, introducing errors in the gadget multiplication may be a problem for the following reason: $\mathtt{mask}(\vec{x}) * \mathtt{mask}(\vec{x}\,')$ equals $DFT_\omega(C(X))$ where C is a degree $2d$ polynomial thus faults on the vector $DFT_\omega(C(X))$ cannot be detected in the $\mathrm{RS}[2d+1, 2d+1]$ code. However, we remark that the first d coefficients of the polynomials involved in $\mathrm{DFT}_\omega(c_{d+1}X^{d+1} + \cdots + c_{2d}X^{2d}) + \mathrm{DFT}_\omega(c_{d+1}X + \cdots + c_{2d-t+1}X^{d-t+1}) * \vec{U} + \sum_{i=1}^{t-1} c_{2d-t+1+i} \cdot G_i$ are null by construction. We deduce that injecting a fault inside these vectors can be detected simply by a syndrome calculation (IDFT). An error may be injected in the coefficient $c_{d+1}, \cdots, c_{2d}$, but in this case the resulting vector $\mathtt{mask}(\vec{x} * \vec{x}\,')$ does not belong to the $\mathrm{RS}[2d+1, d]$ code and the error will be detected. An attacker may inject simultaneously errors in both vectors, but in this case we are no longer in the random injection model and we face an open problem out of scope of this paper.

Finally this leads us to propose below some improvement.

6.5 Detecting faults in the gadget

We propose in this case to slightly modify the parameters of our encoder $x \mapsto (\vec{x}, \vec{r}) \mapsto A^{-1}R$ with $x \in \mathbb{F}_q^t$ and $\vec{r} \in \mathbb{F}_q^{d+1-t}$. We propose to consider some $\vec{r} \in \mathbb{F}_q^{d+1-t-h}$ with $h < d+1-t$. Hence the resulting polynomial has degree $d-h$ instead of d. This modification implies that the vector $\mathtt{mask}(\vec{x}) * \mathtt{mask}(\vec{x}\,') = DFT_\omega(C(X))$ can be checked: $C(X)$ has degree $2d-2h$ in this case and consequently, the vector $DFT_\omega(C(X))$ belongs to the RS$[2d+1, 2d-2h+1, 1+2h]$ code of minimal distance $1+2h$, thus $2h$ errors can be detected. We remind that the error detection on a codeword can be done by computing its syndrome, and computing its syndrome corresponds with our parameters to perform the IDFT algorithm: the computation of $IDFT(\mathtt{mask}(\vec{x}) * \mathtt{mask}(\vec{x}'))$ states whether it corresponds to a degree $d-h$ polynomial or not.

An attacker may inject faults in the vector $\phi(C, \omega)$, however, by construction this vector belongs to RS$[2d+1, 2d-2h+1, 1+2h]$ because $\phi(C,\omega) = \mathtt{mask}(\vec{x}) * \mathtt{mask}(\vec{x}\,') + \mathtt{mask}(\vec{x} * \vec{x}\,')$ and for this error correcting code, up to $2h$ errors can be detected.

Regarding the consequences for the SCA security, the probing order is clearly modified because the dimension of $\vec{r}$ is less than in the original encoder. By analysing carefully the proof of probing order, we observe that this modification does not modify the proof, only the security order is modified, passing from $d-t$ order to $d-t-h$ order. We can now summarize in the following algorithm the step of detection inside the gadget multiplication:

Algorithm 6: `severalByteProduct with detection` Complexity: $n(3+t+4\log(n))$

```
   Input: two vectors $\vec{z} = \mathtt{mask}(\vec{x}) \in \mathbb{F}_q^n$ and $\vec{z}\,' = \mathtt{mask}(\vec{x}\,') \in \mathbb{F}_q^n$
   Output: $\mathtt{mask}(\vec{x} * \vec{x}\,') \in \mathbb{F}_q^n$
1  $\vec{y} \in \mathbb{F}_q^n$
2  for $i \in \{0, \dots, n-1\}$ do
3      $y_i \leftarrow z_i z'_i$
4  $\vec{c}\,'' = \mathtt{ExtractLastCoefficients}(\vec{y}) = (c_{d+h}, \dots, c_{2d}, c_0, \dots, c_{d+h-1})$
5  Check that $(c_{2d-2h+1}, \dots, c_{2d})$ equals the null vector
6  If not, launch a security procedure
7  Else
8  Compute $\vec{y} + \phi(c_{2d-2h+1}, \dots, c_{2d}, \omega)$
9  Check that degree$(\mathrm{IDFT}(\vec{y} + \phi(c_{2d-2h+1}, \dots, c_{2d}, \omega))) \leq d-h$
10 If not, launch a security procedure
11 Else
12 return refresh $(\vec{y} + \phi(c_{2d-2h+1}, \dots, c_{2d}, \omega))$
```

6.5.1 About syndrome computation leakage

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 consider 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 unmasking 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 classic, 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 also, as explained at the beginning of the current section 5.2. 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 we will show later in this section.

In the previous subsection regarding the encoding procedure, we have seen that masking a vector $\vec{x}$ consists in performing

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

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

6.6 Comparison with [BEF$^+$23]

Recently, the authors of [BEF$^+$23] proposed a similar solution based on polynomial encoding. Their solution gives a strong resilience against SCA and simultaneously protects against a huge number of fault injections. We propose to compare the solutions here. We note that our solution works for a fixed length n (number of shares) which is given by the possibility of implementing a DFT instead of multiplying by a Vandermonde matrix whereas their solution has a free length (number of shares) depending on the number of detected errors e: either $n = 2d + e + 1$ in a first version (SotA) or $n = d + e + 1$ for the improved version (laOla). In order to make easier the comparison, we describe our performances with a Vandermonde matrix instead of a DFT and finally, we describe our performances with a trick used for laOla [BEF$^+$23].

Table 4 compiles performance figures and/or complexities of [BEF$^+$23] and our work. This table shows that our scheme is faster, owing to the quasi-linearity complexity of our multiplicative gadget. The difference of complexity also holds for the error detection (and correction) capability, namely quasi-linear in our case *versus* quadratic for [BEF$^+$23]. Moreover, our scheme supports cost amortization, which allows for further speed-up and

Table 4: Comparison between [BEF+23] and our work.

Algorithm	SotA [BEF+23]	This work (genuine, i.e., with DFT)	This work (with Vandermonde matrix)	This work (with DFT and the trick of [BEF+23])	laOla [BEF+23]
Nb of shares	$2d+e+1$	$2d+1$	$2d+e+1$	$2d+1$	$d+e+1$
Cost amort.	No (1)	Yes (t)	Yes (t)	Yes (t)	No (1)
Security order	d	$d+1-t-e/2$	$d+1-t$	$d+1-t$	d
Detected errors	e	e	e	d	e
Amount of randomness in secure multiplication	d^2	$d+1-t$	$d+1-t$	$d+1-t$	d^2
Multiplication gadget complexity	$2d^2+d(e+1)$	$(2d+1)(3+t+4\log(2d+1))$	$2d(d+e+1)$	$3(2d+1)(3+t+4\log(2d+1))$	$3d^2+2d(e+1)$
Error detection (and correction) complexity	$\mathcal{O}(d^2)$	$(2d+1)\log(2d+1)$	$2d(d+e+1)$	$(2d+1)\log(2d+1)$	$\mathcal{O}(d^2)$

huge memory saving. Namely, we can process t sensitive elements altogether whereas [BEF+23] requires to repeat t times the computation.

The only advantage we see for [BEF+23] scheme stems from its flexibility. The fault detection capability can be fine-tuned leveraging the parameter e.

Nonetheless, we attempted to compare our work with [BEF+23] in the context of parametric fault detection capability. In this respect, we had to intentionally degrade our scheme to turn the (quasi-linear) DFT into a (quadratic) multiplication by a Vandermonde matrix. Indeed, DFT is rigid (of fixed size) whereas matrix multiplication is naturally scalable. Despite this handicap, one can notice that our performance are similar (of same quadratic complexity) to that of SotA. Also the error detection (or correction) capability is the same in those conditions. Remarkably, our scheme with "inefficient" Fourier transform still enjoys the advantage to allow for cost amortization.

We note that the authors of [BEF+23] use an extra trick to reduce the degree of the polynomials while $t < d/2$: indeed, we can set:

$$\begin{array}{rcl} P_x(X) & = & \mathrm{IDFT}_\omega(\mathtt{mask}(\vec{x})) \\ & = & P_0(X) + X^{d/2}P_1(X), \\ \text{and} & & \\ P_{x'}(X) & = & \mathrm{IDFT}_\omega(\mathtt{mask}(\vec{x}')) \\ & = & P'_0(X) + X^{d/2}P'_1(X). \end{array}$$

The P_i and $P'i$ can be computed because we have proven in section 6 that the encoder $x \mapsto (x, r)A^{-1}$ is $d+1-t$ probing secure. We have:

$$P_{x'}(X)P_x(X) = P_0(X)P'_0(X) + X^{d/2}\left(P'_0(X)P_1(X) + P_0(X)P'_1(X)\right) + X^d P_1(X)P'_1(X),$$

with:

$$\begin{array}{rcl} T(X) & = & P'_0(X)P_1(X) + P_0(X)P'_1(X) \\ & = & T_0(X) + x^{d/2}T_1(X). \end{array}$$

Then we observe that d errors can be detected on the vectors:

$$\begin{aligned}
\vec{C_0} &= \mathrm{DFT}_\omega(P_0(X)P_0'(X)),\\
\vec{C_1} &= \mathrm{DFT}_\omega(X^{d/2}T_0(X)),\\
\vec{C_2} &= \mathrm{DFT}_\omega(X^{d}T_1(X)), \text{ and}\\
\vec{C_3} &= \mathrm{DFT}_\omega(X^{d}P_1(X)P_1'(X)),
\end{aligned}$$

just by remarking that at least d identified coefficients must be zero for each corresponding polynomial, which enables error detection by syndrom computation.

Finally we underline that our cost amortization capability can be applied for each vectors $\vec{l_i}$, $i \in \{0, 1, 2, 3\}$ in order to get 4 degree d polynomials D_0, D_1, D_2 and D_3 that satisfy $D = D_0 + D_1 + D_2 + D_3$. Hence we avoid the degree $2d$ polynomial in $C(X)$ and consequently, d errors can be detected.

Interestingly, this trick is compliant with our scheme. Thus, our work is also empowered to detect d faults, anywhere in any gadget, where $2d + 1$ is the dimension of the codes. This is reflected in the last-but-one column of Table 4. Our value of the security order benefits from Corollary 1 (i.e., it attains its maximum value $d + 1 - t$), thereby equating the probing security order of [BEF+23] schemes (SotA and laOla).

7 Software implementation

The implementation of a masked AES-128 allowed us to accurately measure the gain in time and memory space that can be obtained with parallel masking (that is, $t > 1$). Indeed, as we can see in Fig. 1, the computation time decreases linearly according to the size of the sensitive data (t), consistently across values d (masking order). We can also witness the quasi-linearity of the computation time (this quasi-affine function depending on the value of d), and the non-linearity (namely, the "quadricity") of RP masking [RP10]:

- the RP masking (in log-log scale) computation time curve grows by *two* decades when d grows by *one* decade,
- whereas for our scheme, the slope is less than two (and the value also is less).

The need for randomness is represented in Fig. 2, and same observations can be done. All values of d are represented for which there exists a DFT (namely $d \in \{1, 2, 7, 8, 25, 42\}$), under the condition $d > t$.

We had to represent speed and randomness for large values of d not because practical applications requires very high masking order, but to show the asymptotic complexity.

We used the C code from Jean-Sébastien Coron's `github` project [Cor] to implement RP. But we replaced the optimized log-table based multiplication by a constant-time one. Namely, hardcoded tables `sq`, `taffine`, `tsmult` in file "`aes_rp.c`" have been replaced by their algorithmic counterparts. The rationale is that masking is pointless if applied on a non-constant time implementation, because timing leakage is exploitable at 1st order [BGV21]. Obviously, we have adopted the same constant-time implementation to our schemes, hence the comparison is fair. Such implementation of field multiplication is used alike in both schemes (RP and ours).

These statistics concern the calculation of 50 times an AES-128 encryption, implemented with C, compiled with `gcc`, with a refresh after each multiplication (SMult) or exponentiation, and executed on an Intel(R) Core(TM) i7-8550U, CPU 1.80 GHz processor, 16 GB of RAM, with different configurations of our scheme compared to Rivain and Prouff (RP) scheme [RP10].

Masking with cost amortization also reduces memory usage. Indeed, with $t = 16$, the total cost to mask a block of 16 bytes is n instead of $16n$. In general, the size of a masked word for AES is $16n/t$.

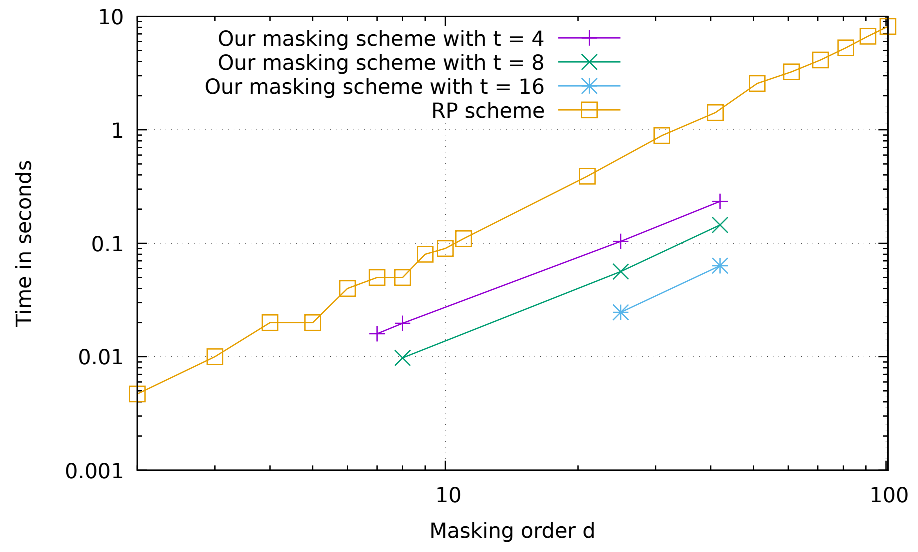

Figure 1: Computation time for 50 times AES calculation, with pre-calculated multiplication.

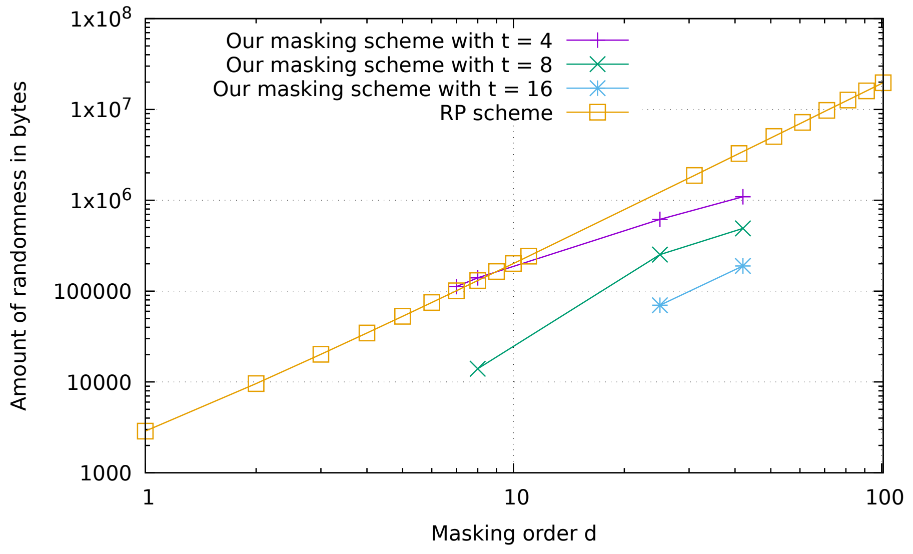

Figure 2: The amount of randomness generated in terms of bytes

8 Limitations and Future Work

Side-channel security order. One drawback of our masking scheme is that the order of masking cannot be freely chosen. Namely n shall divide $q-1$ (recall Sec. 2.1) and the choice of n is further limited by Eqn. (1) (which precludes in particular that $n = q-1$). For instance, for the cases of:

- AES ($q = 256$), the values of n are $\{3, 5, 15, 17, 51, 85\}$, i.e. $d \in \{1, 2, 7, 8, 25, 42\}$ (recall $n = 2d + 1$);
- Crystals Kyber ($q = 3329$), the values of n are $\{2^i, 2 \leq i \leq 8\} \cup \{13 \cdot 2^i, 0 \leq i \leq 7\}$, i.e. $d \in \{2, 4, i8, 16, 32, 64, 128, 6, 13, 26, 52, 104, 208, 416, 832\}$ (note that $n = 2d + 1$ if n is odd but $n = 2d$ if n is even).

9 Conclusions and perspectives

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 concomitantly masking (construction known as cost 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.

We show actual implementation complexity results in software and detail our gain in terms of performance.

As a perspective, we intend to show results in hardware and show the gain of our masking in terms of gate size and power consumption as well.

Acknowledgments

The research of the first author is partly supported by the Norwegian Research Council. The two last authors declare that this work has partly benefited from the funding by French Bank for Innovation (BPI), through the project **X7PQC** (project call "*Cryptographie post quantique*", held by the National Quantum Strategy "Develop the post-quantum cryptographical offering" and the National Cyber Strategy "Development of innovative and critical cyber technologies").

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, *EUROCRYPT*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485. Springer, 2015.

[BEF$^+$23] Sebastian Berndt, Thomas Eisenbarth, Sebastian Faust, Marc Gourjon, Maximilian Orlt, and Okan Seker. Combined fault and leakage resilience: Composability, constructions and compiler. *IACR Cryptol. ePrint Arch.*, page 1143, 2023.

[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.

[BGV21] Antoine Bouvet, Sylvain Guilley, and Lukas Vlasak. First-Order Side-Channel Leakage Analysis of Masked but Asynchronous AES. In Pantelimon Stănică, Sihem Mesnager, and Sumit Kumar Debnath, editors, *Security and Privacy*, pages 16–29, Cham, 2021. Springer International Publishing.

[BHP98] Richard E Blahut, W. Cary Huffman, and Vera 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.

[Cor] Jean-Sébastien Coron. HTable countermeasure against side-channel attacks — reference implementation for the masking scheme presented in [Cor14]. Source code available from: https://github.com/coron/htable.

[Cor14] Jean-Sébastien Coron. Higher Order Masking of Look-Up Tables. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458. Springer, 2014.

[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.

[FRSG22] Jakob Feldtkeller, Jan Richter-Brockmann, Pascal Sasdrich, and Tim Güneysu. CINI MINIS: domain isolation for fault and combined security. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1023–1036. ACM, 2022.

[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, *ASIACRYPT*, 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.

[GPK$^+$21] Michael Gruber, Matthias Probst, Patrick Karl, Thomas Schamberger, Lars Tebelmann, Michael Tempelmeier, and Georg Sigl. Domrep-an orthogonal countermeasure for arbitrary order side-channel and fault attack protection. *IEEE Trans. Inf. Forensics Secur.*, 16:4321–4335, 2021.

[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.

[MAN$^+$19] Lauren De Meyer, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. M&m: Masks and macs against physical attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):25–50, 2019.

[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.

[MZ22] Maria Chiara Molteni and Vittorio Zaccaria. A relation calculus for reasoning about t-probing security. *J. Cryptogr. Eng.*, 12(1):1–14, 2022.

[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.

[Pla22] Maxime Plançon. Exploiting algebraic structures in probing security. Cryptology ePrint Archive, Paper 2022/1540, 2022. https://eprint.iacr.org/2022/1540.

[RMB$^+$18] Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel P. Smart. CAPA: the spirit of beaver against physical attacks. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 121–151. Springer, 2018.

[RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, 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.

[SMG16] Tobias Schneider, Amir Moradi, and Tim Güneysu. ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO*, volume 9815 of *Lecture Notes in Computer Science*, pages 302–332. Springer, 2016.

[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.

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 $n = 3, 5, 15, 17, 51, 85$. 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 `SAGE` code and the executable source code in C language of our implementations are provided in a GitHub: https://github.com/daif-abde/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. The case $n = 5$ is not a power of two but we can propose a decomposition that leads to very low 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. 3.

$$\begin{tikzcd}
 & & X^6+X \arrow[dll] \arrow[dr] & & & \\
X^4+X^3 \arrow[d] & & & X^4+X^3+X^2+X+1 \arrow[dl] \arrow[dr] & & \\
X^2+X \arrow[d] \arrow[dr] & & X^2+(\omega+\omega^4)X+1 \arrow[d] \arrow[dr] & & X^2+(\omega^3+\omega^2)X+1 \arrow[d] \arrow[dr] & \\
X & X+1 & X+\omega & X+\omega^4 & X+\omega^3 & X+\omega^2
\end{tikzcd}$$

Figure 3: 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 in 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 $20 > 6\ln^2(6) > 19$.

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 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:

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

thus, according to [WZ88], we get the following decomposition tree depicted in Fig. 4. 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.

$$\begin{tikzcd}
 & & & X^2+X \arrow[r] \arrow[rd] & X \\
 & & X^4+X \arrow[ru] \arrow[rd] & & X+1 \\
 & & & X^2+X+1 \arrow[r] \arrow[rd] & X+\omega^5 \\
 & X^8+X^4+X^2+X \arrow[ruu] \arrow[rdd] & & & X+\omega^{10} \\
 & & & X^2+X+\omega^5 \arrow[r] \arrow[rd] & X+\omega \\
 & & X^4+X+1 \arrow[ru] \arrow[rd] & & X+\omega^4 \\
 & & & X^2+X+\omega^{10} \arrow[r] \arrow[rd] & X+\omega^2 \\
X^{16}+X \arrow[ruuuu] \arrow[rdddd] & & & & X+\omega^8 \\
 & & & X^2+X+\omega \arrow[r] \arrow[rd] & X+\omega^7 \\
 & & X^4+X+\omega^5 \arrow[ru] \arrow[rd] & & X+\omega^9 \\
 & & & X^2+X+\omega^4 \arrow[r] \arrow[rd] & X+\omega^6 \\
 & X^8+X^4+X^2+X+1 \arrow[ruu] \arrow[rdd] & & & X+\omega^{13} \\
 & & & X^2+X+\omega^2 \arrow[r] \arrow[rd] & X+\omega^3 \\
 & & X^4+X+\omega^{10} \arrow[ru] \arrow[rd] & & X+\omega^{14} \\
 & & & X^2+X+\omega^8 \arrow[r] \arrow[rd] & X+\omega^{11} \\
 & & & & X+\omega^{12}
\end{tikzcd}$$

Figure 4: Polynomial decomposition tree for $X^{16} + X$ on $\mathbb{F}_{256}$.