Securing Lattice-Based KEMs with Code-Based Masking: A Theoretical Approach*

Pierre-Augustin Berthet[1,2][0009−0005−5065−2730] (✉), Cédric Tavernier[2][0009−0007−5224−492X], Jean-Luc Danger[1][0000−0001−5063−7964], and Laurent Sauvage[1][0000−0002−6940−6856]

[1] Télécom Paris, 19 Place Marguerite Perey, F-91123 Palaiseau Cedex, France
{(✉)berthet,jean-luc.danger,laurent.sauvage}@telecom-paris.fr
[2] Hensoldt SAS France, 115 Avenue de Dreux, 78370 Plaisir, France
{pierre-augustin.berthet,cedric.tavernier}@hensoldt.net

Abstract. The recent technological advances in Post-Quantum Cryptography (PQC) raise the questions of robust implementations of new asymmetric cryptographic primitives in today's technology. This is the case for the lattice-based Module Lattice-Key Encapsulation Mechanism (ML-KEM) algorithm which is proposed by the NIST as the first standard for Public Key Encryption (PKE) and Key Encapsulation Mechanism (KEM), taking inspiration from CRYSTALS-Kyber. We have notably to make sure the ML-KEM implementation is resilient against physical attacks like Side-Channel Analysis (SCA) and Fault Injection Attacks (FIA). To reach this goal, we propose to adapt a masking countermeasure, more precisely the generic Direct Sum Masking method (DSM). By taking inspiration of a previous paper on AES, we extend the method to finite fields of characteristic prime other than 2 and even-length codes. We also briefly investigate its application to Keccak, which is the hash-based function used in ML-KEM. We provide the first masked implementation of ML-KEM with both SCA and FIA resilience while not relying on any conversion between different masking methods. Our FIA resilience allows for fault correction even within the multiplicative gadget. Finally, we adapt a polynomial evaluation method to compute masked polynomials with public coefficients over finite fields of characteristic different from 2.

Keywords: Post-Quantum Cryptography · CRYSTALS-Kyber · ML-KEM · FIPS 203 · Side-Channel Analysis · Fault Injection Attack · Masking · Direct Sum Masking · Code-Based Masking

1 Introduction

Since the dawn of cryptology, cryptanalysis has focused on the theoretical background used to perform cryptography. However, since the late 1990s and the publication of Kocher on Side-Channel Analysis [23], physical attacks try to take advantage of leakages or faults within the implementation rather than breaking the algorithm in itself. For this reason, the software and hardware designers of cryptographic primitives have to take into account this threat. The recent Post-Quantum Cryptographic algorithms are particularly targeted as their implementation still requires secure architectures and analysis to make them robust against physical attacks.
Quantum computing is an active research field which progresses monthly and the likelihood of

* Supported by Agence de l'Innovation de Défense, Ministère des Armées

an efficient quantum computer in the coming 30 years is almost certain [25]. Such a computer would be able to break current asymmetric cryptography primitives by taking advantage of the Shor quantum algorithm [35]. In order to assure a continuity in asymmetric cryptography, the NIST has launched a standardization process of PQC in 2016 [12] resulting in an international competition to create the future digital signature, PKE and KEM protocols which must be secure against quantum and classical computer. The end of the third and final round was announced the 5th of July 2022 [1] and 3 signatures and one PKE/KEM were selected while 4 other KEMs are heading for a final round to serve as alternatives in case of a cryptanalysis breakthrough [3]. The selection process focused first on quantum resilience, cost and performance, and then on the algorithm and its implementation. Most of the candidates claimed to be secure against time-based SCA as they provide constant time implementation and no conditional branching depending on sensitive data. But they do not make them secure against power-based SCA, like like Correlation Power Analysis (CPA), and Fault Injection Attacks (FIA). Even more, some of the candidates contains functions that can not be easily secured using generic defenses and will require specific mechanisms to ensure their Side-Channel resilience.

1.1 Background on Masking

One of the most efficient and proven countermeasure against power-based SCA is masking [11]. The core idea is to avoid manipulating the sensitive data but instead *shares* of it that will be reassembled after the computations are done. The shares being a combination of the sensitive data and a number of random variables called *masks*. Thus, an attacker will only observe leakages from the shares and might not be able to recover the secret data in feasible time. The order of masking is determined by the number of independent shares used. A high-order of masking means a better security against differential attacks but it generally comes at the cost of performances and space.

Classical masking involves either arithmetical masking, where the random shares are subtracted or added to the secret, and boolean masking where the random shares are XORed with the secret. Conversions from one type of masking to the other do exist but have to be performed carefully. Here we will use a generic Code-Based Masking scheme called Direct Sum Masking (DSM), introduced by Bringer et al. [8].

In this paper we focus on ML-KEM [28], inspired by CRYSTALS-Kyber [6], a post-quantum PKE/KEM. They have been already several publications on how to mask it on several platforms. Most noticeably, the work from Heinz et al. [19] proposed the first open-source implementation of a masked Kyber on microprocessor while relying on the work of Oder et al. [29] on previous lattice-based primitives. Bos et al. [7] proposed a masked software implementation of Kyber while Bronchain and Cassiers [9] proposed new gadgets for Arithmetic to Boolean (A2B) and B2A conversions and tested them in a open-source masked implementation of CRYSTALS-Kyber for microprocessors. When it comes to other platforms, Fritzmann et al. [15] worked on masking HW/SW codesign. Beckwith et al. [4] worked on a shared FPGA implementation of CRYSTALS-Kyber and CRYSTALS-Dilithium while masking the CRYSTALS-Kyber.

Remark 1. It is important to note that masking at the first order alone is not a sufficient defense. The PhD thesis work of Kalle Ngo [26] and master's thesis of Linus Backlund [3] (both from KTH

[3] One of the KEMs of the 4th round fell victim to such a breakthrough in August 2022, stressing the need for alternative standards and hybridization

Stockholm) proved that novel methods relying on deep-learning were able to thwart attempts of protecting Kyber with first order masking and/or shuffling. Hence, it is important to either mix defense mechanism (shuffling, blinding, hiding...) or use higher order masking. Also note that these attack methods have not been tested yet against Code-Based Masking.

When it comes to SCA and FIA resilient implementations, Pöppelmann and Heinz [20] proposed a combined fault and DPA protection for lattice-based cryptography. However, they only secured the arithmetic parts of the algorithm and were not able to correct faults, only detecting a few. Fault attacks against masked implementations of ML-KEM are a real concern, with work from Delvaux [13] and Kundu et al. [24] successfully using FIA to break masked implementations of ML-KEM on microcontroller. Thus, there is a clear need for solutions with better resilience to combined attacks using SCA and FIA. Our work aims not only to provide such a solution with better error detection, but also to add an error correcting capability that does not yet exist to this day in the state-of-the-art literature.

1.2 Our Contributions

The first major contribution of this paper is related to the use of Code-Based Masking in a finite field of characteristic other than 2, which is more common in asymmetric cryptography compared to symmetric cryptography. In this work, we extend the masking method from [10] and prove it can also be applied to finite fields of characteristic other than 2. We also prove that we are able to correct several faults within the multiplicative gadget.

The second major contribution of this paper is the first application of Code-Based Masking on a post-quantum cryptography primitive, namely ML-KEM. Not only do we propose a new any-order masking method for this algorithm, but we also propose one that has a built-in solution against FIA. There is currently no other solution in the State-of-the-Art capable of providing resilience against both SCA and FIA for the entirety of the algorithm as the only existing solution from Pöppelmann and Heinz [20] only covers the Number Theoretic Transform (NTT) and cannot correct errors. Please also note that we are the firsts to propose a conversion-free masking of ML-KEM.

Finally, a last contribution is the adaptation of the Paterson and Stockmeyer method of evaluating poynomials [30] to the evaluation of masked polynomials in a finite field of characterisitc different from 2. This method aims at reducing the amount of multiplication between two sensitive data when evaluating a polynomial, as such multiplications are noticeably more costly to perform than additions and scalar multiplications. We provide details on the exact complexity of performing such an evaluation in a secure manner.

The paper is structured as follows: in Section 2, we introduce notations and ML-KEM. In Section 3, we present our Direct Sum Masking and our adaptation of the Horner method in Section 3.9. In Section 4, we explain how we adapted our masking method to ML-KEM. Finally, in Section 5, we discuss performances. Section 6 concludes our paper.

2 Preliminaries

2.1 Notations

Let $n \in \mathbb{N}^*$ the length of a code and d its dimension. We denote odm the masking order. We consider the finite field $\mathbb{F}_q$ with q a prime integer. Let ν a primitive element of $\mathbb{F}_q$. We assume that $n \neq 0 \mod q$ divides $q-1$, then we have

$$\omega = \nu^{\frac{q-1}{n}} \Rightarrow \omega^n = 1.$$

We must distinguish the case n odd and n even, then we set $d = \lfloor n/2 \rfloor$. For any vector $(u_0, ..., u_{n-1}) \in \mathbb{F}_q^n$, we can associate the polynomial $U(X) = u_0 + u_1 X + \ldots + u_{n-1} X^{n-1}$ and the discrete Fourier transform is defined by

$$\mathtt{DFT}_\omega(u_0, ..., u_{n-1}) = \left(\sum_{i=0}^{n-1} u_i \omega^{ij} \right)_{j \in [0 \ldots n-1]} = \left(U(\omega^j) \right)_{j \in [0 \cdots n-1]}.$$

Then the $\mathtt{DFT}_\omega$ inverse is defined by:

$$\mathtt{IDFT}_\omega(U(1), \ldots, U(\omega^{n-1})) = n^{-1} \left(\sum_{i=0}^{n-1} U(\omega^i) \omega^{-ij} \right)_{j \in [0 \cdots n-1]} = (u_0, ..., u_{n-1}).$$

Remark 2. We have clearly made the hypothesis "n divides $q-1$" to find the condition of application of the Fast Fourier Transform but the procedure that we are going to develop obviously works by considering respectively $\mathtt{DFT}_\omega$ and $\mathtt{IDFT}_\omega$ as a Vandermonde multiplication and its inverse. The impact is just in term of complexity which cost n^2 multiplications over $\mathbb{F}_q$ against $\mathcal{O}(n \log n)$ for a $\mathtt{DFT}_\omega$ in the most favourable cases.

For the ML-KEM algorithm, the considered $\mathtt{DFT}_\omega$ is coming from methods described in [10,37]. It consists in building a tree (see section 4) of polynomials and to compute input vector interpreted as a polynomial modulo these polynomials. In particular cases, for example over finite fields of even characteristic and $n+1$ a power of two, the tree is composed of linearized polynomials (up to constant) which are sparse by nature [37]. For example in the figure below, the tree is defined over the finite field $\mathbb{F}_{2^4}$ with $n = 3$ and ω satisfying $\omega^3 = 1$:

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

Then, to calculate $\mathtt{DFT}_\omega(C)$ with $C = (c_0, c_1, c_2, c_3)$ and $C(X) = c_0 + c_1 X + c_2 X^2 + c_3 X^3$, we first compute $C_{1,0} = C(X) \mod q_{1,0}$ and $C_{1,1} = C(X) \mod q_{1,1}$, finally we get the result by performing $C_{1,0} \mod q_{0,0}$, $C_{1,0} \mod q_{0,1}$, $C_{1,1} \mod q_{0,2}$ and $C_{1,1} \mod q_{0,3}$. We show in the section 4 that we have the same principles with the ML-KEM parameters.

We have seen that the $\mathtt{DFT}_\omega$ operation is equivalent to a Vandermonde matrix multiplication $V(\omega)$ with $V(\omega) = (\omega^{ij})_{i,j \in [\![0, 2d-1]\!]}$ and

$$\mathtt{DFT}_\omega(u_0, ..., u_{n-1}) = (u_0, ..., u_{n-1}) \times V(\omega).$$

For length n vectors of the form $(u_0, ..., u_{d-1}, 0, \dots, 0)$, the $\mathtt{DFT}_\omega$ operation corresponds to an encoding procedure by the Reed-Solomon code denoted: $\mathrm{RS}[n, d, n-d+1]$. A generator matrix of this code is given by the shortened matrix $(\omega^{ij})_{i\in[\![0,d-1]\!], j\in[\![0,2d-1]\!]}$. We recall some results that can be found in [31]: This error correcting code is classic, it is a MDS (maximum distance separable) code, which means that its minimal distance is optimal and equals $n-d+1$ where n is code length and d is its dimension. Among the good properties of these codes, we have, if R is a generator matrix of MDS code $\mathcal{R}$ of length n and dimension d that:

- If $\mathcal{R}$ is MDS, then $\mathcal{R}^\perp$ is MDS where $\mathcal{R}^\perp$ is the code defined by kernel(R);
- If R is MDS, then all set of d columns are free.

We recall that any $[n, d, n-d+1]$-linear code can detect until $n-d$ errors.

2.2 ML-KEM

ML-KEM or FIPS 203 [28] is the first post-quantum KEM standard by the NIST. It is a slight modification of CRYSTALS-Kyber [2,6], a Module-Lattice-Based KEM which has been selected at the end of the 3^{rd} round of the NIST Standards Post-Quantum Competition in July 2022 [1]. It relies on several instances of the Module-LWE/LWR problems for its key generation, encapsulation and decapsulation procedures.
At its core, ML-KEM is a CPA-Secure PKE. To ensure CCA-level of security and a KEM status, a modified version of the Fujisaki-Okamoto Transform [16] is used.
ML-KEM has three levels of security, with different parameter sets (see [28] Table 2 page 33 or Table 4 in Appendix B). All sets use the same modulo, namely $q = 3329$. We also denote $\mathbb{Z}_q[X]/(X^{256}+1)$ by R_q and $S_\eta := \{P \in R_q, \|P\|_\infty \leq \eta\}$ a subset of R_q.
Amongst other notations defined by ML-KEM, we have $\lceil \rfloor$, the nearest integer with ties rounded up used in the compression functions, defined as follow:

$$Compress_q(\alpha, d_i) = \left\lceil \frac{2^{d_i}}{q} \cdot \alpha \right\rfloor \, mod \; 2^{d_i}, \alpha \in \mathbb{Z}_q \tag{1}$$

$$Decompress_q(\beta, d_i) = \left\lceil \frac{q}{2^{d_i}} \cdot \beta \right\rfloor, \beta \in \mathbb{F}_{2^{d_i}} \tag{2}$$

When applied to a vector of polynomials, those two functions will then be applied to each coefficient of each polynomial separately.

Remark 3. It is interesting to note that, for $d_i = 1$, the $Decompress_q$ function can be seen as a simple multiplication by a scalar, as the value β in the equation 2 can be extracted from the rounding as it can only be 0 or 1. Thus, we have $\lceil \frac{q}{2} \cdot \beta \rfloor = 1665 \cdot \beta$. This does not apply to $Compress_q$ (Equation 1) however.

Remark 4. It is also important to note that the compression functions are lossy:

$$\text{If } m' = Decompress_q(Compress_q(m, d_i), d_i), \text{ then } |m - m'| \leq \lceil q/2^{d_i+1} \rfloor \tag{3}$$

In ML-KEM, the distribution used for random sampling of sensitive values is the Center Binomial Distribution:

$$CBD_\eta(\beta) = \sum_{i=0}^{255} (\sum_{j=0}^{\eta} \beta_{2i\eta+j} - \sum_{j=0}^{\eta} \beta_{2i\eta+\eta+j}) X^i \text{ with } \beta \in \{0,1\}^{512\eta} \tag{4}$$

This function is fed with a pseudo-random input β, generated by

$$PRF(seed, N) = SHAKE256(seed \| N)$$

The counter here allows seed reuse for the multiple values sampled during the PKE algorithms of ML-KEM. We will use the $\hookleftarrow$ notation for sensitive value sampling. Keep in mind this is a call to Equation 4 where the input is $PRF(seed, N)$. The N counter is incremented after each call to CBD.
Non sensitive values are sampled a bit differently but this is out of the scope of this paper and we will simply denote this sampling by $\leftharpoonup$.

We will only present the KEM Decapsulation of ML-KEM here. For more details, we invite you to consult Appendix B where are described the PKE and KEM algorithms as well as figures showing the sensitiveness of the different operations within ML-KEM. You can also consult the reference paper of ML-KEM [28] for the algorithms and [32] (slide 76), [34] (slide 32-35) for the sensitiveness.

Algorithm 1 KEM Decapsulation

Input: Ciphertext $c = (c_u, c_v)$
Input: Secret Key $sk = (\vec{s}, pk, h = H(pk), z)$
Output: Shared key K
$\vec{u}, v = Decompress_q(c_u, d_u), Decompress_q(c_v, d_v)$
$m' := PKE.Decrypt(\vec{s}, \vec{u}, v)$ ▷ (see Appendix B, Figure 5 or [28])
$(K', seed') := G(m' \| h)$
$\tilde{K} = J(z \| c, 32)$
$\vec{u'}, v' := PKE.Encrypt(pk, m', seed')$ ▷ (see Appendix B, Figure 4 or [28])
$c' = (Compress_q(\vec{u'}, d_u), Compress_q(v', d_v))$
if $c \neq c'$ **then**
    $K' = \tilde{K}$
end if
return K'

Remark 5. H, G and J are all different Keccak instances.

Remark 6. Keep in mind that PKE.Encrypt will always result in the same outputs for a given set of inputs, as the seed for the sampling is one of the inputs. Thus, tampering with the ciphertexts results in tampering with the seed and a completely different result out of the re-encapsulation.

If you are interested in knowing more about ML-KEM, we invite you to read the FIPS 203 (draft) standard from the NIST [28] and the CRYSTALS-Kyber specification papers [2,6].

3 Code-Based Masking, a DSM Example

The DSM encoding [8] consists in mapping the information x in a masked information (x, r) where r is a random mask such that:

$$x \mapsto (x, r) \mapsto x\mathbf{G} + r\mathbf{H}, \tag{5}$$

where $\mathbf{G}$ and $\mathbf{H}$ are two generator matrices of the two complementary codes $\mathcal{C}$ and $\mathcal{D}$ with $\mathcal{C} \cap \mathcal{D} = \{0\}$.

We propose to describe a masking method based on Reed Solomon encoding. This method is described in [10] for the characteristic 2 and odd length. We show in this section that it works for the characteristic prime q. We want to mask an information of size t and we assume that $\omega \in \mathbb{F}_q$ is a n-square root of unity and we consider a free family $u_0, u_1, u_2, ..., u_{d-1}$ of $\mathbb{F}_q^{d-1}$ with $u_i \neq \omega^j$ for any $0 < i \leq t-1$ and $0 < j \leq n-1$. We want now to mask the vector $\vec{x} = (x_0, \ldots, x_{t-1}) \in \mathbb{F}_q^t$ with $t < d$ and $d = \lfloor n/2 \rfloor$.

3.1 Encoding Procedure

First we pick randomly $\vec{r} = (r_t, r_{t+1}, \ldots, r_{d-1})$ in $\mathbb{F}_q^{d-t}$. It is well known that there exist a vector $\vec{a} = (a_0, a_1, \ldots, a_{d-1})$ and the associate polynomial $P_{\vec{x}}(X) = a_0 + a_1 X + \cdots + a_{d-1} X^{d-1}$ of degree at most $d-1$ that satisfies $P_{\vec{x}}(u_i) = x_i$ for $i \in \{0, \ldots, t-1\}$ and $P_{\vec{r}}(u_i) = r_i$ for $i \in \{t, \ldots, d-1\}$.

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

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

The second step of our masking procedure consists in evaluating the $P_{\vec{r}}$ over the set $1, \omega, \omega^2, \ldots, \omega^{n-1}$. By construction, the second step of encoding consists in computing $\mathtt{DFT}_\omega(a_0, \ldots, a_{d-1}, 0, \ldots, 0)$. Thus finally:

$$\mathtt{Mask}(\vec{x}) = \mathtt{DFT}_\omega(a_0, \ldots, a_{d-1}, 0, \ldots, 0).$$

Algorithm 2 `SeveralByteMasking` Complexity : d^2

1: **Input:** a sensitive vector $\vec{x} \in \mathbb{F}_q^t$
2: **Output:** $\mathtt{Mask}(\vec{x}) \in \mathbb{F}_q^n$
3: $\vec{r} \xleftarrow{\$} \mathbb{F}_q^{d-t}$
4: $\vec{a} \leftarrow (\vec{x} \mid \vec{r}) \times A^{-1}$ ▷ A^{-1} is a precomputed value
5: **return** $\mathtt{DFT}_\omega(\vec{a} \mid \vec{0})$

We have presented a $\mathcal{O}(d^2)$ complexity encoding procedure, but we can do better with the following method: We can construct $P(X) = T_t(X) + R_t(X)$ by first picking randomly the polynomial $T_t(X) = a_t X^t + \cdots + a_{d-1} X^{d-1}$. Then we evaluate T_t over $1, u, \ldots, u_{t-1}$ which cost $t(d-1-t)$ multiplications over $\mathbb{F}_q^{d-1}$. We want now constructing $R_t(X) = a_0 + a_1 X + \ldots, a_{t-1} X^{t-1}$ which leads to solve the linear system

$$\underbrace{\begin{bmatrix} 1 & u_0 & \ldots & u_0^{t-1} \\ & & \vdots & \\ 1 & u_i & \ldots & u_i^{t-1} \\ & & \vdots & \\ 1 & u_{t-1} & \ldots & u_{(t-1)}^{t-1} \end{bmatrix}}_{A^\top} \times \underbrace{\begin{bmatrix} a_0 \\ \vdots \\ a_i \\ \vdots \\ a_{t-1} \end{bmatrix}}_{\vec{a}\,'} = \underbrace{\begin{bmatrix} x_0 + T_t(u_0) \\ \vdots \\ x_i + T_t(u_i) \\ \vdots \\ x_{t-1} + T_t(u_{t-1}) \end{bmatrix}}_{\vec{y}\,'} .$$

The matrix inversion of A is a precomputation, thus, the calculation of:

$$\vec{a}\,' = (A^{-1})^{\top}\vec{y}\,'$$

costs $(t+1)^2$ multiplications over $\mathbb{F}_q$. Hence, the total cost of this encoding (including the $T_t(u_i)$ calculation) does not exceed $t(d-1-t)+t^2 = t(d-1)$ multiplications over $\mathbb{F}_q$. Again, the second step of encoding consists in computing $\mathtt{DFT}_\omega(a_0, \ldots, a_{d-1}, 0, \ldots, 0)$ which can be achieved with not more than $(2d-1)\log(2d-1)$ multiplications over $\mathbb{F}_q$.

Remark 7. All the aforementioned operations are obviously reversible and we denote by `Unmask` the reverse operation. A tedious calculation gives a complexity in $t(d-1)+(2d-1)\log(2d-1)$ multiplications over $\mathbb{F}_q$.

3.2 Error Correcting Code Interpretation

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

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

We note that this `DFT` computation corresponds to the encoding in the Reed-Solomon code defined by the evaluation of $1, X, \ldots, X^{d-1}$ over $1, \omega, \omega^2, ..., \omega^{n-1}$, and represented by a Vandermonde matrix $V(\omega)$. Hence, we get that

$$\mathtt{Mask}(\vec{x}) = (\vec{x}, \vec{r})R^{\top}V(\omega) \quad (= \vec{x}G + \vec{r}H \text{ in the DSM model}).$$

We deduce that our masking algorithm corresponds to encoding procedure with a generalized Reed-Solomon code of minimal distance $n-d+1$, dimension d and length n.

3.3 Masking Addition, Subtraction and Scaling

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

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

3.4 Masking the Multiplication

Let's denote: $\vec{z} = \mathtt{Mask}(\vec{x})$ and $\vec{z}\,' = \mathtt{Mask}(\vec{x}')$. Obviously,

$$\vec{z} \odot \vec{z}\,' = \mathtt{DFT}_\omega(a_0, \ldots, a_{d-1}, 0, \ldots, 0) \odot \mathtt{DFT}_\omega(a'_0, \ldots, a'_{d-1}, 0, \ldots, 0).$$

The polynomial obtained by performing $\texttt{DFT}_\omega^{-1}(\texttt{DFT}_\omega(P_{\vec{x}}) \times \texttt{DFT}_\omega(P_{\vec{x}'})) = P_{\vec{x}}(X) \times P_{\vec{x}'}(X) = C(X)$ is a $2d-2$ degree polynomial, which satisfies $C(u_i) = P_{\vec{x}}(u_i) \times P_{\vec{x}'}(u_i) = x_i x'_i$ for i in $\{0, \dots, t-1\}$.
Now we have to propose a method that associates a degree $d-1$ polynomial $D(X)$ to $C(X)$. This polynomial must satisfies the same properties: $D(u_i) = C(u_i)$ for all $0 \le i \le t-1$.
The authors of [17] proposed the following construction for $t = 1$:

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

Obviously, in this case $D(u_0) = C(u_0) = x_0 x'_0$. This construction can be generalized and let:

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

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

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

The degree $d-1$ polynomial $D(X)$ satisfies $D(u_i) = C(u_i) = x_i x'_i$ of $i \in \{0, \dots, t-1\}$.
In order to build efficiently $\texttt{DFT}_\omega(D(X))$, let's write:

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

Thus:

$$\begin{aligned} \texttt{DFT}_\omega(D(X)) = \quad & \texttt{DFT}_\omega(C(X)) \\ & - \texttt{DFT}_\omega(c_d X^d + \cdots + c_{2d-2} X^{2d-2}) \\ & + \texttt{DFT}_\omega(c_d X + \cdots + c_{2d-t-1} X^{d-t}) \odot \vec{u} \\ & + \textstyle\sum_{i=1}^{t-1} c_{2d-t-1+i} \cdot G_i. \\ = \quad & \texttt{Mask}(\vec{x} \odot \vec{x}') \ , \end{aligned}$$

where: $G_i = \texttt{DFT}_\omega(\sum_{j=1}^{t} U_j(X) u_j^{d-t+i})$ for $i \in \{1, \dots, t-1\}$ and $\vec{u} = \texttt{DFT}_\omega(\sum_{j=1}^{t} U_j(X))$ are a precomputed values, and $c_d, \dots, c_{2d-2} = \texttt{extractLastCoefficients}(\vec{z} \odot \vec{z}\,')$. We remind that *extractLastCoefficients* has been defined in [10]:

We have seen that $\texttt{IDFT}_\omega(\vec{z} \odot \vec{z}\,') = (c_i)_{i \in \{0,\dots,n-1\}} = C(X)$, then if we denote $\vec{y} = \vec{z} \odot \vec{z}\,'$, by definition $c_{j+d} = \sum_{i=0}^{n-1} y_i \omega^{-i(j+d)} = \sum_{i=0}^{n-1} (y_i \omega^{-id}) \omega^{-ij} \ \forall\ 0 \le j \le d-1$ and $(c_{j+d})_{j \in \{0,\dots,d-1\}}$ is obtained from $\texttt{IDFT}\left((y_i \omega^{-id})_{0 \le i \le n-1}\right)$.

If we denote $\phi(C,\omega) = -\mathtt{DFT}_\omega(c_{d+1}X^d + \cdots + c_{2d-2}X^{2d-2}) + \mathtt{DFT}_\omega(c_dX + \cdots + c_{2d-t-1}X^{d-t}) \odot \vec{u} + \sum_{i=1}^{t-1} c_{2d-t-1+i} \cdot G_i$ where C represents the $d-1$ last coefficients of $\mathtt{IDFT}(\mathtt{Mask}(\vec{x}) \odot \mathtt{Mask}(\vec{x}'))$, then we get that

$$\mathtt{Mask}(\vec{x} \odot \vec{x}') = \mathtt{Mask}(\vec{x}) \odot \mathtt{Mask}(\vec{x}') + \phi(C,\omega)$$

3.5 Security Proof

One way of proving the security of an implementation is through a theoretical approach using formal security models. Such a model for SCA resilience is the d-probing security model, first introduced by Ishai et al. [21]. To prove the security of our design in this model, we will first introduce some definitions.

Definition 1. *[36] (Private circuit compiler [21]). A private circuit compiler for a circuit C with input in $\mathbb{F}_q^{\phi}$ and output in $\mathbb{F}_q^{\phi'}$ is defined by a triple (I,T,O) where*

- *$I : \mathbb{F}_q^{\phi} \to \mathbb{F}_q^{\varphi}$ is an input encoder that randomly maps the input in $\mathbb{F}_q^{\phi}$ to the input sharing in $\mathbb{F}_q^{\varphi}$.*
- *T is a circuit transformation whose input is circuit C, and output is a randomized circuit C', whose input is the input sharing $\mathbb{F}_q^{\varphi}$, and the output in $\mathbb{F}_q^{\varphi'}$ is called outputsharing.*
- *$O : \mathbb{F}_q^{\varphi'} \to \mathbb{F}_q^{\phi'}$ is a decoder that maps the output sharing in $\mathbb{F}_q^{\varphi'}$ to the output of C in $\mathbb{F}_q^{\phi'}$.*

We say that (I,T,O) is a private circuit compiler and C' is a $d-$private circuit (or $d-$probing secure, where d is called the security order) if the following requirements hold:

- *Correctness: for any input $a \in \mathbb{F}_q^{\phi}$, $PrO(C'(I(a))) = C(a) = 1$.*
- *Privacy: for any input $a \in \mathbb{F}_q^{\phi}$ and any set of probes $\mathcal{P}$ such that $|\mathcal{P}| \le d$, $C'_{\mathcal{P}}(I(a))$ are independent of the input a.*

Definition 2. *[36]* (Encoder, Codeword, Sharing, Valid Sharing and Share)*. An encoder, Enc : $\mathbb{F}_{q^k} \to \mathbb{F}_{q^n}$ is a probabilistic algorithm that maps a vector in $\mathbb{F}_{q^k}$ to a vector in $\mathbb{F}_{q^n}$. The latter vector in $\mathbb{F}_{q^n}$ is called codeword or valid sharing. A sharing is a vector in $\mathbb{F}_{q^n}$, and the elements of a codeword or sharing are called shares. Moreover, an encoder is called $d-$private encoder if and only if the joint distribution of any d shares are independent of the input of the encoder, where the probability is over the random coins from the encoder.*

We propose to show in the following paragraphs that our method corresponds to $(d-t)$-probing order for the security with a discussion around more sophisticate security models. For fault injection resilience, we assume that we are in the random fault model with a reasonable number of injected faults.

SCA Resilience We showed that this construction is identical to the original construction of [10] up to the sign and up to the parity of n. The proof is coming from the property of this masking that can be written as a DSM encoding [8]:

$$x \mapsto (x,r) \mapsto xG + rH.$$

For this model, the masking order is provided by the minimal distance of the code $H^{\perp}$. It is proven in [10] that the probing order depends of code $H^{\perp}$ which can be MDS (i.e $d_{min} = d+2-t$)

or AMDS (i.e $d_{min} = d + 1 - t$). We show in the subsection 4 (with $t = 1$) that we are in the MDS favourable case. It means that odm equals $d + 1 - t$, i.e $d_{min} = d + 2 - t$. The gadget multiplication is also in this case $(d+1-t)$-probing secure due to the MDS property of H. Let's remind the main steps of the proof:
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 $H = (0, Id_{d-t})A^{-1}R$.

Proposition 1. *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 $\text{rank}(H) = d-t$. 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 [36], a direct consequence is that the encoding procedure enc_B is d'-private. Our task consists now in evaluating d'.

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

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

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*

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

Proof. We denote by K the matrix which corresponds to the last $d-t$ rows of A^{-1}, then

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

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

By construction, the parity check matrix of R that we can denote T is a generator matrix of the 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 $K = (0_t, Id_{d-t})A^{-1}$. 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$[d, d]$ code. We would like to determine now the dual code of K and we observe the equation $A^{-1} \times A = Id_d$. By setting

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

we get that

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

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

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

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

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:

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

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

We deduce that $R_{d\times n} \times Ri_{n\times d} = Id_d$ with $R = R_{d\times n}$. The matrix V_ω^{-1} is Vandermonde matrix associated to $IDFT_\omega$, then $R_i = Ri_{n\times d} = (\omega^{-ij})_{i\in[\![0..n-1]\!], j\in[\![0..d-1]\!]}$. 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$[n, d]$ code and $D = {}^tB \times {}^tR_i$.

We note that ${}^tR_i = (\omega^{(n-i-1)j})_{i\in[\![0..d-1]\!], j\in[\![0..n-1]\!]}$ is a generator matrix of a code generated by d polynomials of degree more than $n - d$. Then ${}^tB = (u_i^j)_{i\in[\![0..t-1]\!], j\in[\![0..d-1]\!]}$. Hence the code generated by D is an evaluation code generated by t independent polynomials of degree more than d whereas T is a generator matrix of a code generated by $d - 1$ 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

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

Let

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

and

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

Then

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

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

$$\frac{u_0^d X^{n-d} - X^{n-2d}}{u_0 - X} \tag{6}$$

over $\{1, \omega, \ldots, \omega^{n-1}\}$ and we are looking for a degree $d-1$ polynomial $P(X)$ that cancels the maximum of positions of D_0, i.e. such that $Q(X) = (X - u_0)P(X) - X^{n-2d} + u_0^d X^{n-d}$ admits the maximum of zeros. We remark that degree$(Q) \le d$, then the number of zero is less than $n - d$ which is equivalent to a minimal distance greater than d. In the same time, the Singleton bound states that $d_{\min}(T \cup D_0) \le d + 1$. We deduce that for $D = D_0$,

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

For $t = 2$, the Singleton bound states that $d_{\min}(T \cup D_0 \cup D_1) \le d + 2 - t$. 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

$$P(X) = \frac{u_0^d X^{n-d} - X^{n-2d}}{u_0 - X} + \theta \frac{u_1^d X^{n-d} - X^{n-2d}}{u_1 - X}.$$

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

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

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

Gadget multiplication resilience The steps of the proof can be summarized by the hypothesis that has be done in [18] and proven in [10]:

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 $t_n^{DFT} \ge d - t$.

Proof. In fact the application $DFT_\omega(\vec{x}\|0) \mapsto r$ corresponds exactly to our masking operation $\mathtt{Mask}(\vec{x}) = (\vec{x}, \vec{r}) \times A^{-1} \times R$ 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 $\mathtt{DFT}_\omega^{-1} : u' \mapsto tt$: if fact, $u' = refresh(\mathtt{Mask}(\vec{x}) * \mathtt{Mask}(\vec{y}))$ where $*$ represents here the multiplication term by term and not the mask multiplication. In our masking, by definition, we have $u' = \mathtt{Mask}(\vec{0}) + \mathtt{Mask}(\vec{x}) * \mathtt{Mask}(\vec{y})$. $\mathtt{Mask}(\vec{0}) = \vec{r}H$ 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, $\mathtt{DFT}_\omega^{-1}(\mathtt{Mask}(\vec{0})) = (a_0(r), a_1(r), \ldots, a_d(r), 0, \ldots, 0) = (0, r)A^{-1}$. Then $\mathtt{DFT}_\omega^{-1}(\mathtt{Mask}(\vec{x}) * \mathtt{Mask}(\vec{y})) = (c_0, \ldots, c_{2d})$. 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, the following lemma is deduced in [18]:

Lemma 1. *[18] The circuit processing* $(\mathtt{Mask}(x), \mathtt{Mask}(y)) \mapsto u = \mathtt{Mask}(x) * \mathtt{Mask}(y)$ *is at least* $(d-t)$*-probing secure.*

For simplicity, we denote $d_n =$To conclude about the security of the gadget multiplication, we remind that the Mask multiplication (gadget) is obtained from the following computation:

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

where $G_i = \mathtt{DFT}_\omega(\sum_{j=0}^{t-1} U_j(X) u_j^{d-t+i})$ 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 $\mathtt{DFT}_\omega(c_d X^d + \cdots + c_{2d-2} X^{2d-2})$, $\mathtt{DFT}_\omega\left((c_d X + \cdots + c_{2d-t-1} X^{d-t}) * \vec{U}\right)$ and $\sum_{i=1}^{t-1} c_{2d-t-1+i} \cdot G_i$ involves only the variables $c_d, \ldots, c_{2d-t-1}$ related to the sensitive information. Hence, the weakest side is obtained with the vector

$$(c_d, \ldots, c_{2d-2}) = \mathtt{ExtractLastCoefficients}(\vec{z} * \vec{z}').$$

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 secure, 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 $(c_d, \ldots, c_{2d-2})$, $x = \mathtt{unmask}(z)$ and $x' = \mathtt{unmask}(z')$ cannot be reconstructed: if an attacker has access to the following system of equations

$$\begin{cases} c_{2d-2} &= a_{d-1}a'_{d-1} \\ c_{2d-3} &= a_{d-2}a'_{d-1} + a_{d-1}a'_{d-2} \\ c_{2d-4} &= a_{d-3}a'_{d-1} + a'_{d-3}a_{d-1} + a'_{d-2}a_{d-2} \\ &\vdots \\ c_{2d-2-k} &= \sum_{i=0}^{k} a_{d-i-1}a'_{d-(k-i)-1} \\ &\vdots \\ c_d &= \sum_{i=0}^{d-1} a_{d-i-1}a'_{i+1}. \end{cases}$$

We can evaluate the number of potential solutions for $(a_i)_{i\in[\![0..d-1]\!]}$: by assuming that $c_{2d} \neq 0$, then the equation $c_{2d-2} = a_{d-1}a'_{d-1}$ admits $q-1$ solutions. If $c_{2d} = 0$, then $a_d a'_d$ admits q solutions. By setting $a_d \neq 0$ and $a'_d \neq 0$ we get the equation $c_{2d-3} = a_{d-2}a'_{d-1} + a_{d-1}a'_{d-2}$ admits q solutions. By induction, we get the same property at any step $k \leq d-1$. Thus totally this system admits at least $q \times q^{1+2+\ldots+d-2} > q^{1+\frac{(d-2)^2}{2}}$ solutions for the set of d variables a_i. Hence, this system of equation does not give information about the a_i with $i \in [\![0..d-1]\!]$. By symmetry, we get the same property for a'_i, but we know according the proof of the theorem 1 that we must get at least $d-t$ values of a_i to expect reconstructing the encoded sensitive information, thus we conclude that the gadget multiplication is at least $d-t$ probing secured.

Remark 8. It seems that our encoding method has similar properties than this one defined in [18] then it would be interesting to investigate if the region probing security still holds here.

Faults Injection By construction, everywhere a codeword C is present, the integrity of C can be checked by computing the syndrome of C, i.e by computing $\texttt{IDFT}(C) = c$ and checking that c corresponds to a degree $d-1$ polynomial. If not, it means that some errors have been introduced. According our parameters, C belongs to the Reed-Solomon code $\mathrm{RS}[2d, d, d+1]$ and can detect d errors.

The difficult question concerns the gadget multiplication between two vectors $\texttt{Mask}(x)$ and $\texttt{Mask}(y)$. For this computation we must perform $\texttt{Mask}(x) \odot \texttt{Mask}(y)$ where "$\odot$" corresponds to the multiplication term by term. We showed that $\texttt{Mask}(x) \odot \texttt{Mask}(y) = \texttt{DFT}_\omega(C(X))$ where $C(X)$ is a degree $2d-2$ polynomial. However, our codewords have length $n = 2d$ and $\texttt{DFT}_\omega(C(X)) \in \mathrm{RS}[2d, 2d-1, 2]$. Hence we can check with a syndrome calculation (i.e $\texttt{IDFT}(\texttt{Mask}(x) \odot \texttt{Mask}(y))$) that $C(X)$ is degree $2d-2$ polynomial. If not, it means that at least one error has been injected. Then an attacker may inject faults on the vector $(c_d, \ldots, c_{2d-2})$, however in this case we remind that

$$\texttt{Mask}(\vec{x} \odot \vec{y}) = \texttt{Mask}(\vec{x}) \odot \texttt{Mask}(\vec{y}) + \phi((c_d, \ldots, c_{2d-2}), \omega),$$

with $\texttt{Mask}(\vec{x} \odot \vec{y})$ and $\texttt{Mask}(\vec{x}) \odot \texttt{Mask}(\vec{y})$ that can be verified, then the injected fault will be detected.

We showed here that our gadget supports one fault injection. As shown in [10], to support more injections we could modify our encoder by reducing the dimension of $\vec{r}$. As a direct consequence, the degree of the resulting polynomial $C(X)$ from a multiplication has a degree strictly less than $2d-2$ and more errors can be detected. In the same time this modification decrease the security probing order, thus, it is a question of balance. However, a trick used in [5] to reduce the degree of the polynomials which is compliant with our scheme while the number of information symbols $t < d/2$: indeed, we can set $P_x(X) = \texttt{IDFT}_\omega(\texttt{Mask}(\vec{x})) = P_0(X) + X^{d/2}P_1(X)$ and $P_{x'}(X) = \texttt{IDFT}_\omega(\texttt{Mask}(\vec{x}')) = P'_0(X) + X^{d/2}P'_1(X)$. The P_i and P'_i can be computed because 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 $T(X) = P'_0(X)P_1(X) + P_0(X)P'_1(X) = T_0(X) + x^{d/2}T_1(X)$. Then we observe that d errors can be detected on the vectors $\vec{C}_0 = \texttt{DFT}_\omega(P_0(X)P'_0(X))$, $\vec{C}_1 = \texttt{DFT}_\omega(X^{d/2}T_0(X))$, $\vec{C}_2 = \texttt{DFT}_\omega(X^dT_1(X))$ and $\vec{C}_3 = \texttt{DFT}_\omega(X^dP_1(X)P'_1(X))$, just by remarking that at least d identified coefficients must be null for each corresponding polynomial. Finally our cost amortization trick can be applied for each vectors $\vec{C}_i$, $i \in \{0, 1, 2, 3\}$ in order to get 4 degree d polynomials D_0, D_1, D_2 and D_3 that satisfies $D = D_0 + D_1 + D_2 + D_3$. Hence we avoid the degree $2d$ polynomial in $C(X)$ and consequently, d errors can be detected by applying our detection method.

Remark 9. We assume that faults are random and do not directly affect the syndrome computations that detect them. We could also assume that a tamper-resistant component performs the fault detection task.

3.6 Complexity

It is shown in [10] that the complexity of the multiplication is quasi-linear as it requires $\mathcal{O}(4d\log(2d))$ multiplications in $\mathbb{F}_q$. This is a standard complexity, but regarding real performances and applicability a study must be performed over different platforms (hardware and software) with different strategies: parallel computation, pipeline, bitslicing... From now on, we set $t = 1$ as it seems us difficult to take benefit of several symbol encoding due to the design of CRYSTALS-Kyber. Taking $t > 1$ may be interesting if we manage to compute simultaneously several KEM computation but it affects the probing security order.

In terms of randomness, we require $d - t$ random symbols to mask t sensitive ones. As the multiplication includes a refresh done by adding the mask of $\vec{0}$, it requires another batch of $d - t$ random symbols.

3.7 Masking a Polynomial Function

By induction, we can compute $\texttt{Mask}(x^n)$ for an arbitrary n value in $\mathbb{Z}$. We can write $\texttt{Mask}(x^n) = \texttt{Mask}(x^{n-1} * x) = \texttt{Mask}(x) \odot \texttt{Mask}(x^{n-1}) + \phi(C, \omega)$ thus if we assume that $\texttt{Mask}(x^{n-1})$ has been computed, then the property is demonstrated. The same proof holds for Horner (polynomial evaluation) algorithm.

3.8 Masking a Formal Polynomial

Let $s(X), u(X) \in (\mathbb{F}_q^{(deg)}[X])^2$. We define

$$\texttt{Mask}(s(X)) = \sum_{i=0}^{deg} \texttt{Mask}(s_i)X^i \tag{7}$$

Also, we have $\texttt{Mask}(s(X)) \odot \texttt{Mask}(u(X)) = \texttt{Mask}(s(X) * u(X))$. We deduce that we can perform $\texttt{Mask}(s(X) * u(X))$ using the Karatsuba algorithm of complexity $deg^{1.585}$ `Mask` multiplications [22].

Remark 10. We note that Fast Fourier Transform based on Cooley-Turkey algorithm that involves a n-root of unity could be obviously applied since the scalar multiplication is well defined over the linear codes. We preferred Karatsuba because this part of CRYSTALS-Kyber algorithm is not the most costly in term of masking.

3.9 Adapting the Horner method to masked polynomials

The Horner method evaluates a polynomial of degree d in d multiplications and d additions. However, masked multiplications between two sensitive data, that we will denote by "sensitive multiplications", are costly. On the other hand, masked multiplications between a sensible data and a public one are cheap as they can be considered as scaling (see Section 3.3). We take inspiration from the work of Paterson and Stockmeyer [30] where they perform a variation of the Horner method. We adapt it to the specific case of masked evaluation of public polynomials in characteristic different from 2. Please note that characteristic 2 has already been covered by Roy and Vivek in [33].

Theorem 3. *Let P be a public polynomial and d its degree. Let's assume first that $s = \sqrt{d}$ is an integer for simplicity. We notice that $P(X)$, where X is the sensitive data we mask, can be written as:*

$$\begin{gathered} P(X) = P_0(X) + X^s P_1(X) + \ldots + X^{d-s} P_{s-1}(X), \\ \textit{with } \forall i \in [\![0, s-2]\!], \deg(P_i(X)) \leq s-1 \textit{ and } \deg(P_{s-1}(X)) = s \end{gathered} \tag{8}$$

The complexity of evaluating this polynomial in X is $2s-2$ sensitive multiplications, $(s-1)^2$ scalar multiplications and $d-s$ additions.

Proof.

1. *Sensitive multiplications.* First we must evaluate all the X^j up to $j = s$. They will be re-used inside all the P_i. Thus, we have $s-1$ sensitive multiplications to perform. Once it is done for the P_i, we can consider the polynomial in Equation 8 as a polynomial of degree $s-1$ in $Y = X^s$. Thus, by applying the normal Horner method, it will require only $s-1$ sensitive multiplications. For a total of $(s-1)+(s-1) = 2s-2$.
2. *Scalar multiplications.* $\forall i \in [\![0, s-2]\!], \deg(P_i(X)) \leq s-1$. Thus, we have $(s-1)*(s-2)$ scalar multiplications to perform. For P_{s-1}, we have a polynomial of degree s and thus $s-1$ scalar multiplications. For a total of $(s-1)(s-2)+(s-1) = (s-1)^2$.
3. *Additions* Regarding the evaluation of the P_i, there is as much additions as scalar multiplications, so $(s-1)^2$. However, we must add to that count the additions performed during the final Horner on $Y = X^s$. For a total of $(s-1)^2 + (s-1) = d-s$.

Remark 11. If d is not a square, let e be the greatest square lower than d and $r = \sqrt{e}$. The complexity of evaluating P becomes $2r-1$ sensitive multiplications, $r(r-2)+(d-e) = d-2r$ scalar multiplications and $(d-2r)+r = d-r$ additions.

3.10 Masking Boolean Operations

We proved that we can use our masking method whenever arithmetical operations are used. However, some algorithms requires boolean operations. Lets recall some simple properties working for $x, y \in \{0,1\}^2$:

$$x \wedge y = x * y, \ x \oplus y = x + y - 2 * x * y$$

Thus, in the very specific case where the value before masking is equal to 0 or 1, we are able to perform basic boolean operations while in masked state by using these arithmetic equations.

3.11 Masking Keccak

The current standard for hash functions is FIPS-202 [14], also known as Keccak or SHA3. It can be seen as a 3-dimensional array denoted by A of size $(5, 5, w)$. At its core is the `Round` function and its 5 steps. All the operations within Keccak are either boolean operations or permutations within the state array.

We adopted a simple strategy when it comes to masking Keccak. We choose to stay within our DSM over $\mathbb{F}_q$. To do so we will mask Keccak by converting each of its operands to its masked counterpart over $\mathbb{F}_q$. We will also mask each bit separately instead of working on 64-bits words.

Remark 12. Current state of the art regarding masking ML-KEM [7,9] use conversions between different masking methods before and after the use of Keccak in order to benefit from a fast boolean masked implementation of Keccak. However, converting is costly and can create vulnerabilities. For instance, the work from Bronchain and Cassiers [9] was a patch for leakage in the conversion used in [7]. Kundu et al. [24] successfully targets the conversion with a fault attack to recover sensitive data. Thus, using conversion between masking method can be risky.
To the best of our knowledge there is currently no work on the process of converting between two different Code-Based Maskings. Our attempts at a fully SCA and FIA resilient Code-Based Masking conversion gadget proved too costly and completely outweights the advantage given by the use of a boolean code to mask Keccak. We plan on publishing our work on the conversion in a follow-up paper.

For more details on the Keccak function, please refer to the standard paper from the NIST [14].

4 ML-KEM Example

4.1 Discussion on Parameters

We have several possibilities when it comes to which code we can use and for specific masking orders. In this section, we will propose several examples of parameters for different masking orders.

First Order of Masking We propose to consider the parameters $n = 4 = 2d'$, $d = 2$, $\omega = \nu^{\frac{q-1}{4}} = 1729$ and $\alpha = \nu^{\frac{q-1}{13}} = 2970$. We chose to these parameters,

$$A = \begin{pmatrix} 1 & 1 \\ \alpha & \alpha^2 \end{pmatrix}, \ V(\omega) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \end{pmatrix}$$

Then,

$$\texttt{Mask}(x) = (x, r) \times (A^{-1}) \times V(\omega),$$

with $r \in \mathbb{Z}_q$ is picken randomly.

$$\texttt{Mask}(x) = (x, r) \begin{pmatrix} 103 & 2590 & 1545 & 2387 \\ 3227 & 740 & 1785 & 943 \end{pmatrix} = xG + rH,$$

It is possible to check (cf: MAGMA online) that the minimal distance of $H^{\perp}$ is 2 as predicted by the theory, then $odm = 1$. Furthermore $V(\omega)$ is a Reed Solomon code and 2 faults can be detected. The complexity of the detection corresponds to complexity of the syndrome computation that can be achieved with a DFT and for $n = 4$ we have the following tree decomposition:

$$\begin{tikzcd}
 & & & X^4 - 1 \arrow[dll] \arrow[drr] & & & \\
 & X^2 - 1 \arrow[dl] \arrow[dr] & & & & X^2 + 1 \arrow[dl] \arrow[dr] & \\
X - 1 \arrow[d] & & X + 1 \arrow[d] & & X + 1600 \arrow[d] & & X + 1729 \arrow[d] \\
P(1) & & P(-1) & & P(-1600) & & P(-1729)
\end{tikzcd}$$

It is shown in [37] that this representation is favourable to hardware implementation and complexity does not exceed $nlog(n)$ multiplications over $\mathbb{F}_q$.

Remark 13. The current state-of-the-art [20] on combined SCA and FIA resilient implementations is able to mask at the first order[4] and to detect at most 1 fault for the lowest parameter and 3 for a higher parameter at the cost of performance. Our method masks at the first order and detects 2 faults. We improve the state-of-the-art by being able to correct 1 fault. We also propose higher orders of masking.

Third Order of Masking For higher order, we make the following choice: $n = 8 = 2d'$, $d = 4$, $\omega = \nu^{\frac{q-1}{8}} = 749$ and $\alpha = \nu^{\frac{q-1}{13}} = 2970$. We chose to these parameters,

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 2970 & 2379 & 1492 & 341 \\ 2379 & 341 & 2292 & 3095 \\ 1492 & 2292 & 781 & 1812 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 \\ \alpha^3 & \alpha^8 & \alpha^9 & \alpha^{12} \end{pmatrix}, \; V(\omega) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \omega^4 & \omega^5 & \omega^6 & \omega^7 \\ 1 & \omega^2 & \omega^4 & \omega^6 & 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega & \omega^4 & \omega^7 & \omega^2 & \omega^5 \end{pmatrix}$$

Then,

$$\texttt{Mask}(x) = (x, r_1, r_2, r_3) \times A^{-1} \times V(\omega),$$

and

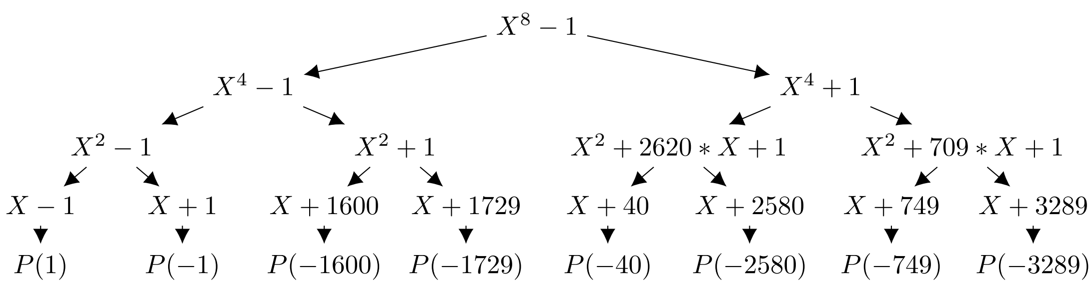
$$\texttt{Mask}(x) = (x, r_1, r_2, r_3) \times \begin{pmatrix} 3212 & 747 & 3112 & 897 & 1801 & 1931 & 428 & 1649 \\ 1893 & 2178 & 3029 & 3078 & 1546 & 491 & 3239 & 631 \\ 2127 & 2027 & 2130 & 1062 & 1354 & 3312 & 3206 & 2416 \\ 2756 & 1707 & 1717 & 1622 & 1958 & 925 & 3115 & 1963 \end{pmatrix},$$

[4] Their method is not masking per say but it offers the same level of protection than first order masking

with $\vec{r} = (r_1, r_2, r_3) \in \mathbb{Z}_q$ is picken randomly.

$$\texttt{Mask}(x) = xG + \vec{r}H, \text{ with } H = \begin{pmatrix} 1893 & 2178 & 3029 & 3078 & 1546 & 491 & 3239 & 631 \\ 2127 & 2027 & 2130 & 1062 & 1354 & 3312 & 3206 & 2416 \\ 2756 & 1707 & 1717 & 1622 & 1958 & 925 & 3115 & 1963 \end{pmatrix}.$$

We have a minimal distance of $H^{\perp}$ equal to 4 and consequently, $odm = 3$. We can detect 4 faults. These parameters lead also to a very fast DFT with the following tree decomposition:



Remark 14. We present the first and third order of masking in this paper. While second order is possible, the code chosen to do so will not have the same complexity when it comes to detecting faults as we will not be able to use the DFT for the syndrome computation.

Higher Orders of Masking As stated in Section 2.2, ML-KEM operations are defined over $\mathbb{Z}_q$ with $q = 3329$ satisfying $q - 1 = 2^8 \times 13$. If $\nu = 3$ is a primitive element of $\mathbb{F}_q$, we could set $n = 13 = 2d' + 1$, $d = 6$, with $\omega = \nu^{\frac{q-1}{13}}$ and $\alpha = \nu^{\frac{q-1}{16}}$. The masking method is working for these parameters but we have not found a better way to compute the $\texttt{DFT}$ than using a Vandermonde matrix multiplication which costs $\mathcal{O}(n^2)$. However, with these parameters, for $t = 1$, we get $odm = d + 1 - 1 = 6$ and 6 faults can be detected on codewords.

Code-Based Masking offers a high flexibility in terms of which code we can use. We can choose either to improve the FIA resilience or the SCA resilience.

4.2 Masking Strategy

First we will focus on securing the PKE.Decrypt part of ML-KEM (see [28] Algorithm 14 page 28 or Figure 5 in Appendix B). Then we will discuss how we can extend our masking method to the entirety of the KEM Decapsulation procedure (see Algorithm 1) and ML-KEM itself.

You can use the following map (Figure 1) of the KEM Decapsulation to navigate between the different parts we had to mask.

Graph legend:

- : Non-sensitive operation
- : Non-sensitive input/output of the algorithm
- : Non-sensitive intermediate data
- : Sensitive operation
- : Sensitive input/output of the algorithm

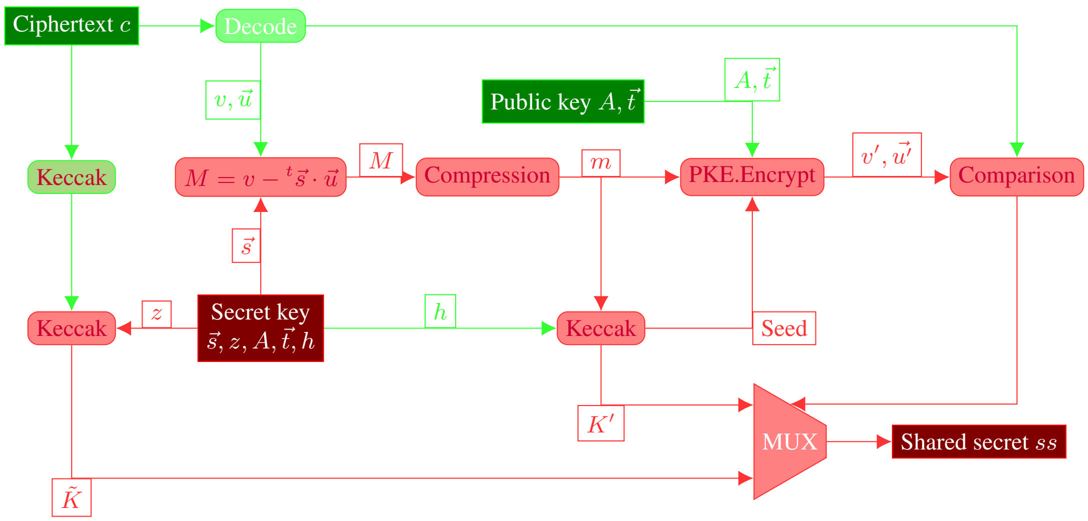


Fig. 1. Interactive map (links) of our masking strategy

– ———▶ : Sensitive intermediate data

Remark 15. We choose to add a hash function to hash the ciphertext before using it with z in $\tilde{K} = J(z\|c, 32)$. The reason is performances. By prehashing the ciphertext, we significantly reduce the size of the input of a masked Keccak while the prehashing is done on a public data and thus with a non-masked Keccak. This modification only affects the output of KEM Decapsulation in the event of a FO Transform failure. This does not affect the theoretical security of ML-KEM and we still are fully compatible with other implementations of ML-KEM.

We propose to mask the following operation: $v - {}^t\vec{s} \cdot \vec{u}$ with $v \in R_q$ public, $\vec{u} \in R_q^{sec}$ public, and $\vec{s} \in R_q^{sec}$ secret. First, we have to discuss how to multiply a sensitive data and a public one.

Karatsuba between a Sensitive and a Public Polynomials To multiply two sensitive polynomials we choose to rely on the Karatsuba algorithm. However, most of the multiplications between two polynomials in ML-KEM involve a sensitive and a public one. To avoid the cost of masking a public polynomial, we will instead consider its coefficients as scalars. Indeed, we have the following theorem:

Theorem 4. *Let $f(X) \in R_q$ be a sensitive data and $g(X) \in R_q$ a public one. Then* $\texttt{Mask}(f(X) * g(X)) = \texttt{Mask}(f(X)) * g(X)$

Proof. First we will prove it for a degree lesser than 16 and then recursively extend the theorem to the entirety of the Karatsuba algorithm as for this degree, we use the textbook polynomial. The textbook polynomial multiplication states that

$$\left(\sum_{i=0}^{15} f_i X^i\right) * \sum_{j=0}^{15} g_j X^j = \sum_{j=0}^{15} g_j * \left(\sum_{i=0}^{15} f_i X^{i+j}\right) \tag{9}$$

Using the linearity of our masking method, it's immediate that

$$\texttt{Mask}(\sum_{j=0}^{15} g_j * \left(\sum_{i=0}^{15} f_i X^{i+j}\right)) = \sum_{j=0}^{15} g_j * \left(\sum_{i=0}^{15} \texttt{Mask}(f_i) X^{i+j}\right) \tag{10}$$

Now, let's recall the Karatsuba formula:

$$\begin{aligned} f(X) * g(X) &= (f'(X) + f''(X)X^{n/2}) * (g'(X) + g''(X)X^{n/2}) \\ &= f' * g'X^n + (f' * g' + f'' * g'' + (f' + f'') * (g' + g''))X^{n/2} + f'' * g'' \end{aligned}$$

Each product is once again between a sensitive polynomial f and a public one g. Thus, we have to prove that $\texttt{Mask}(f'(X)*g'(X)) = \texttt{Mask}(f'(X))*g'(X)$(same for $f''*g''$ and $(f'+f'')*(g'+g'')$) at degree j. If true, then using the linearity it'll also be true for $f(X)*g(X)$ at degree $2j$. However, we already proved it for degree lesser than 16 in Equation 10. By recursion we proved that we do not need to mask non sensitive data before multiplying them with sensitive polynomials as we can see their coefficients as scalars.

Thanks to the Theorem 4 and the homomorphic properties of the `Mask` procedure we have:

$$\texttt{Mask}(v - {}^t\vec{s} \cdot \vec{u}) = \texttt{Mask}(v) - \sum_{i=0}^{sec} \texttt{Mask}(s_i) * u_i,$$

where ${}^t\vec{s} \cdot \vec{u} = \sum_{i=0}^{sec}(s_i * u_i)$ with $s_i \in R_q$ and $u_i \in R_q$.

Compression The next step is to apply the $Compress_q$ function while staying masked. We have the following theorem:

Theorem 5. *$Compress_q$ can be computed using a polynomial function.*

Proof. We can rewrite the $Compress_q$ function from Equation 1 as

$$\forall \alpha \in \mathbb{Z}_q,\ Compress_q(\alpha) = \begin{cases} 1 & \text{if } \lceil \frac{q}{4} \rfloor < \alpha < \lceil \frac{3q}{4} \rfloor \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

As $\mathbb{Z}_q$ is a finite field, we can simply enumerate all the values of α resulting in 1 and those resulting in 0 and thus we can rewrite Equation 11 as

$$\forall \alpha \in \mathbb{Z}_q,\ Compress_q(\alpha) = \begin{cases} 1 & \text{if } \alpha \in \{\alpha_0, \alpha_1, \dots, \alpha_h\} \\ 0 & \text{if } \alpha \in \{\beta_0, \beta_1, \dots, \beta_l\} \end{cases} \text{ with } h + l = q \tag{12}$$

A simple Lagrange interpolation of Equation 12 thus give us the following:

$$\forall \alpha \in \mathbb{Z}_q,\ Compress_q(\alpha) \Leftrightarrow P(X = \alpha) = \prod_{i=0}^{l}(X - \beta_i) \sum_{j=0}^{h} \prod_{k=0, k\neq j}^{h} \frac{(X - \alpha_k)}{(\alpha_j - \alpha_k)} \tag{13}$$

We proved $Compress_q$ can be seen as a polynomial function and we can further extend the reach of our masking method within ML-KEM.

We deduce from this theorem that we can mask $Compress_q$ as a polynomial function. Evaluating this polynomial has a complexity of $2\lfloor\sqrt{3328}\rfloor - 1 = 113$ masked multiplications using the method from Section 3.9. However, this polynomial has a structure. All its exponents are odd. Thus, by setting $Y = X^2$ we can gain some complexity and evaluate this polynomial in $2\lfloor\sqrt{1665}\rfloor - 1 + 1 = 80$ masked multiplications.

Using the adaptation of the Paterson-Stockmeyer [30] method as in Section 3.9, we are able to perform the masked message compression in just 80 sensitive multiplications instead of the 3328 required by a straight application of the Horner method.

From PKE to KEM PKE Decrypt is used in KEM Decapsulation as shown in Algorithm 1. In order to mask the rest of the KEM Decapsulation procedure of ML-KEM and the other KEM procedures, we have to address a few points.

– **How do we hash the message output of PKE Decrypt?** The message is a polynomial where each term is either 0 or 1 masked. As stated in Subsection 3.11, we need the input to be masked bits, which is the case here. Thus, we can directly apply our DSM Keccak implementation on the output of PKE Decrypt.

– **How do we mask PKE Encrypt?** The homomorphic properties of `Mask` can also be applied to both the PKE encryption and key generation of ML-KEM. By taking into account Remark 3 regarding the $Decompress_q$ function, we can secure most of the computations using the sensitive data $\vec{s}, m, \vec{r}, \vec{e}, \vec{e_1}$ and e_2 in PKE Key Gen (see [28] page 26 Algorithm 12) and PKE Encrypt (see [28] page 27 Algorithm 13 or Figure 5 Appendix B). For instance, to compute v we do

$$\mathtt{Mask}(v) = \left(\sum_{i=0}^{sec} t_i * \mathtt{Mask}(r_i)\right) + \mathtt{Mask}(e_2) + 1665 * \mathtt{Mask}(m) \tag{14}$$

However, we have to secure the sampling of these sensitive data.

– **How do we use our masking method to perform sensitive data sampling?** To sample sensitive values in the PKE Encrypt procedure, we use the CBD from Equation 4 fed by two chained Keccak instances. As the output of our masked Keccak implementation is a vector of either 0s or 1s masked, it can be fed into a new masked Keccak implementation without problems.
We can compute the following:

$$\mathtt{Mask}(CBD_\eta(G)) = \sum_{i=0}^{255}(\sum_{j=0}^{\eta}\mathtt{Mask}(G_{2i\eta+j}) - \sum_{j=0}^{\eta}\mathtt{Mask}(G_{2i\eta+\eta+j}))X^i \tag{15}$$

The result will be masked with our method and ready for use. We can perform sensitive data sampling from a masked seed and remain in the masked domain all along.

Remark 16. Some points discussed here also apply to the sampling of the message in the KEM Encapsulation procedure. Using a TRNG and masking its output, we can have a masked message from the start and thus compute the seed used in the CBD while always staying masked.

- **How do we compare ciphertexts in the Fujisaki-Okamoto Transform without unmasking them?** One of the biggest issue with masking ML-KEM is the lossy nature of the $Compress_q$ function, as stated in Remark 4. As the ciphertext in the KEM Decapsulation (Algorithm 1) is given as input in a compressed state, the NIST draft paper [28] simply compresses the generated ciphertext into c' and compares it with the input ciphertext c. However, we have already seen that masking the $Compress_q$ function can be costly. Thus, papers masking CRYSTALS-Kyber [7,9] use a different approach: They compare the generated ciphertexts $\vec{u'}, v'$ with the decompression of c. This can also be applied to ML-KEM. We went a step further and relied on the property stated in Remark 4 Equation 3. A key point here is we want a function that returns 0 when the ciphertexts are good and not 0 when the comparison fails. Which means that, instead of performing a Lagrange interpolation[5] like for the message compression, here we can just list the values of $y = x - x'$ such as $|y| \leq \lceil q/2^{d_i+1} \rfloor \bmod q$ and consider them as the roots of the polynomial we are looking for. Thus, for $d_i = d_u = 10$, we have $\lceil q/2^{d_i+1} \rfloor = \lceil q/2^{10+1} \rfloor = 2$, thus $y \in [\![-2, 2]\!]$, 5 roots and a polynomial of degree 5:

$$P(X) = X^5 + 3324 * X^3 + 4 * X = X * (Y - 4) * (Y - 1) \text{ with } Y = X^2 \tag{16}$$

  This Equation 16 only requires 3 masked multiplications. For $d_i = d_v = 4$, we have $\lceil q/2^{d_i+1} \rfloor = \lceil q/2^{4+1} \rfloor = 104$, thus $y \in [\![-104, 104]\!]$, 209 roots and a polynomial of degree 209. However, we know that X will be a factor of this polynomial and that we will be able to use the symmetric nature of the set of roots to have $(X - a) * (X + a) = X^2 - a^2$, thus allowing us to have two polynomials, X and one of degree 104 in $Y = X^2$. By applying our adaption of the Paterson-Stockmeyer method, we can evaluate this polynomial in just 20 masked multiplications.

We demonstrated that our masking method can be applied completely to ML-KEM[6] to secure computations on sensitive data, without requiring any conversion to a different masking method and providing error detection and error correcting capabilities.

5 Implementation and Performances

5.1 *Proof of Concept* Implementation for ML-KEM

We made a *Proof of Concept* (*PoC*) implementation of our masking method in the C language on a desktop. The intent of this *PoC* is to validate the feasibility of our masking method at an algorithmical level. However, we think it is important to share some performances[7] and results to give an idea of the costs of our method and to highlight the effectiveness of some of our rationales. Figure 2 gives the performances of our masked implementations of each of the three key algorithms of ML-KEM.

---

[5] All the mathematical optimizations in this paper were computed using PARI GP.

[6] Note that we mask ML-KEM-512 but our method works for other security levels as well.

[7] Averages in milliseconds over 1000 iterations.

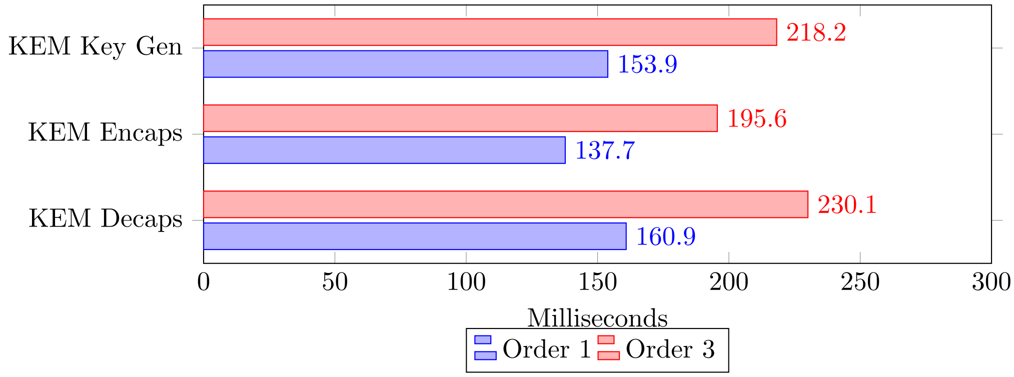


Fig. 2. Performances in milliseconds of our masked ML-KEM-512 at different masking orders

Table 1 shows the performances of our solutions to the several obstacles encountered while masking ML-KEM.

Table 1. Performances in milliseconds of several important functions

Masking order	1	3
Ciphertexts comparison	0.713	1.01
Message compression	4.88	8.01
Hash function	17.1	24.3

5.2 On the Impact of Keccak and Ways to Mitigate It

Our Keccak implementation eats up most of the performances of our *PoC*, as shown in Figure 3. As we already discussed in Section 3.11, trying to convert to a better Code-Based Masking for Keccak did not worked well as the conversion proved too costly. However, there might be another solution we have yet to explore.

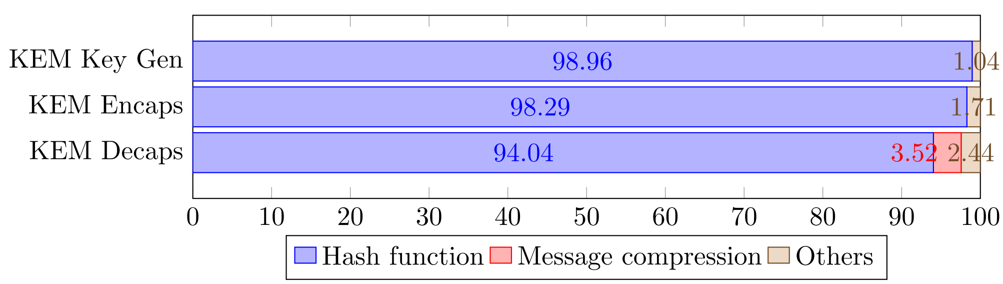


Fig. 3. Impact of Keccak and message compression in percent of time in third order masking

A interesting property of DSM and thus our Code-Based Masking is the possibility of masking several sensitive data within a single mask. This method was first introduced in [36] by Wang et al. under the name of Cost-Amortization and is used in [10]. Applied to ML-KEM, it could help to somehow "parallelize" the different CBD instances and thus reduce the amount of conversions we are forced to perform. It is important to underline that ML-KEM requires more conversions from a boolean Code-Based Masking towards the one we present in this work than the other way around. While we require 256 conversions from our Code-Based Masking towards a boolean Code-Based Masking, the number of inverted conversions is up to 24× higher. This is caused by the outputs of Keccak in the CBD being bigger than their inputs and several CBD instances reusing partially the same input from an earlier conversion. We plan to investigate this solution in a follow-up paper. Alternatively, one could consider an external tamper resistant component to perform the Keccak operations.

5.3 Fast Evaluation of Masked Polynomials

The next Table 2 highlights the efficiency of our adaptation of the Paterson-Stockmeyer method [30] compared to a simple application of the Horner method or the use of the structure of the message compression polynomial. Indeed, it can be factorized in several smaller polynomials. As the biggest irreducible factor is of degree 599, this sets an evaluation complexity of slightly more than 600 sensitive multiplications. Please note that all those evaluations use the variable change $Y = X^2$ and thus are evaluated in Y.

Table 2. Impact of several evaluation methods on the message compression in milliseconds

Masking order	1	3
Message compression using the Horner method	37.25	56.65
Message compression using factorisation	12.77	17.68
Message compression using the Paterson-Stockmeyer method	4.88	8.01

Those experimental results clearly validate the theoretical results we highlighted in Section 3.9.

5.4 Comparison with previous works

To the best of our knowledge, there is no other paper proposing a masked implementation of ML-KEM on a personal computer we could compare to. We intend to port our method on a FPGA platform for experimental validation in a future work. This would allow us to compare our work with [20]. Keep in mind that the goal of this *PoC* is to prove the algorithmical feasibility of our masking method. This paper aims at laying the theoretical foundations on which future works and implementations on better targets (Micro-controller, FPGA, ASIC...) will be able to rely on.

Nonetheless, we compared our results with the reference C source code of CRYSTALS-Kyber in Table 3:

Table 3. Performances factors compared to CRYSTALS-Kyber reference implementation

Masking order	0 (reference time)	1	3
KEM Key Gen	0.03 ms	×5130	×7273
KEM Encapsulation	0.03 ms	×4590	×6520
KEM Decapsulation	0.03 ms	×5363	×7670
Keccak	0.0007 ms	×24 429	×34 714

The impact of our Keccak implementation is once again highly noticeable.

The performances shown in this Section 5 were realized on a laptop computer equipped with a 11th Gen Intel(R) Core(TM) i7-11850H processor operating at 2.50 GHz with 16 GB of RAM. The source code was compiled and executed using gcc version 11.3.0. A particularity of our setup is the use of Ubuntu 22.04.1 through WSL2 (Windows Subsystems for Linux) on a computer operating Windows 11.

Remark 17. Due to the intellectual property regulation in place in our working environment, we are not able to share the source code of our *PoC* for the moment.

6 Conclusion and Future Work

In this paper we proved in Section 3 that Code-Based masking can be used with finite fields of prime characteristic other than 2 and with codes of even length. We proposed an optimal method to evaluate a masked polynomial in characteristic different from 2 by using a adaptation of the Paterson-Stockmeyer [30] method in Section 3.9. We demonstrated the first masked implementation of a post-quantum KEM using the Code-Based Masking method in Section 4. We succeeded in proposing a masked implementation of ML-KEM where sensitive data are masked once and never require any conversion or unmasking. We also provide a better security against Fault Injection Attacks (FIA) compared to the current state-of-the-art [20] by not only being able to detect several faults but also to correct some, even within the gadgets, which is an improvement on the DSM method proposed in [36].

As stated several time in this work, we intend to present a follow-up paper discussing the conversion between two Code-Based Masking methods and the use of the cost-amortization method to further improve the performances of our solution. The next step of our work will be to implement this solution on a FPGA hardware platform and verify its robustness experimentally. This will also ease the comparison with state-of-the-art and future implementations. As for other asymmetric cryptography primitives, we expect our method to work on ML-DSA [27] with some minor tweaks, as ML-DSA and ML-KEM have a lot in common. Our work on enabling this masking method for finite fields of characteristic different from 2 should also allow us to further explore solutions with resilience to both SCA and FIA for current pre-quantum cryptography primitives.

Acknowledgments This work was realized thanks to the grant 2022156 from the Appel à projets 2022 thèses AID Cifre-Défense by the Agence de l'Innovation de Défense (AID), Ministère des Armées (French Ministry of Defense). This paper is also part of the on-going work of Hensoldt SAS France for the Appel à projets Cryptographie Post-Quantique launched by Bpifrance for the Stratégie Nationale Cyber (France National Cyber Strategy) and Stratégie Nationale Quantique (France National Quantum Strategy). In this, Hensoldt SAS France is a part of the X7-PQC project in partnership with Secure-IC, Télécom Paris and Xlim.

References

1. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., et al.: Status report on the third round of the nist post-quantum cryptography standardization process. US Department of Commerce, NIST (2022). https://doi.org/10.6028/NIST.IR.8413-upd1
2. Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber algorithm specifications and supporting documentation. pq-crystals (2021)
3. Backlund, L.: A side-channel attack on masked and shuffled implementations of m-lwe and m-lwr cryptography: A case study of kyber and saber (2023)
4. Beckwith, L., Abdulgadir, A., Azarderakhsh, R.: A flexible shared hardware accelerator for nist-recommended algorithms crystals-kyber and crystals-dilithium with sca protection. In: Cryptographers' Track at the RSA Conference. pp. 469–490. Springer (2023). https://doi.org/10.1007/978-3-031-30872-7_18
5. Berndt, S., Eisenbarth, T., Faust, S., Gourjon, M., Orlt, M., Seker, O.: Combined fault and leakage resilience: Composability, constructions and compiler. Cryptology ePrint Archive, Paper 2023/1143 (2023), https://eprint.iacr.org/2023/1143
6. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber: a cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018). https://doi.org/10.1109/EuroSP.2018.00032
7. Bos, J.W., Gourjon, M., Renes, J., Schneider, T., Van Vredendaal, C.: Masking kyber: First-and higher-order implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 173–214 (2021). https://doi.org/10.46586/tches.v2021.i4.173-214
8. Bringer, J., Carlet, C., Chabanne, H., Guilley, S., Maghrebi, H.: Orthogonal direct sum masking: A smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks. In: IFIP International Workshop on Information Security Theory and Practice. pp. 40–56. Springer (2014). https://doi.org/10.1007/978-3-662-43826-8_4
9. Bronchain, O., Cassiers, G.: Bitslicing arithmetic/boolean masking conversions for fun and profit: with application to lattice-based kems. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 553–588 (2022). https://doi.org/10.46586/tches.v2022.i4.553-588
10. Carlet, C., Daif, A., Guilley, S., Tavernier, C.: Quasi-linear masking against sca and fia, with cost amortization. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(1), 398–432 (2024). https://doi.org/10.46586/tches.v2024.i1.398-432
11. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19. pp. 398–412. Springer (1999). https://doi.org/10.1007/3-540-48405-1_26
12. Chen, L., Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R.A., Smith-Tone, D.: Report on post-quantum cryptography, vol. 12. US Department of Commerce, National Institute of Standards and Technology . . . (2016)
13. Delvaux, J.: Roulette: A diverse family of feasible fault attacks on masked kyber. Cryptology ePrint Archive, Paper 2021/1622 (2021), https://eprint.iacr.org/2021/1622
14. Dworkin, M.J.: Sha-3 standard: Permutation-based hash and extendable-output functions. NIST FIPS (2015). https://doi.org/10.6028/NIST.FIPS.202
15. Fritzmann, T., Van Beirendonck, M., Roy, D.B., Karl, P., Schamberger, T., Verbauwhede, I., Sigl, G.: Masked accelerators and instruction set extensions for post-quantum cryptography. Cryptology ePrint Archive (2021). https://doi.org/10.46586/tches.v2022.i1.414-460
16. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual international cryptology conference. pp. 537–554. Springer (1999). https://doi.org/10.1007/3-540-48405-1_34

17. Goudarzi, D., Joux, A., Rivain, M.: How to securely compute with noisy leakage in quasilinear complexity. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 547–574. Springer (2018). https://doi.org/10.1007/978-3-030-03329-3_19
18. Goudarzi, D., Prest, T., Rivain, M., Vergnaud, D.: Probing security through input-output separation and revisited quasilinear masking. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 599–640 (2021). https://doi.org/10.46586/tches.v2021.i3.599-640
19. Heinz, D., Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, D.: First-order masked kyber on arm cortex-m4. Cryptology ePrint Archive, Paper 2022/058 (2022), https://eprint.iacr.org/2022/058
20. Heinz, D., Pöppelmann, T.: Combined fault and dpa protection for lattice-based cryptography. IEEE Transactions on Computers **72**(4), 1055–1066 (2022). https://doi.org/10.1109/TC.2022.3197073
21. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23. pp. 463–481. Springer (2003). https://doi.org/10.1007/978-3-540-45146-4_27
22. Karatsuba, A.: Multiplication of multidigit numbers on automata. In: Soviet physics doklady. vol. 7, pp. 595–596 (1963)
23. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16. pp. 104–113. Springer (1996). https://doi.org/10.1007/3-540-68697-5_9
24. Kundu, S., Chowdhury, S., Saha, S., Karmakar, A., Mukhopadhyay, D., Verbauwhede, I.: Carry your fault: A fault propagation attack on side-channel protected lwe-based kem. Cryptology ePrint Archive, Paper 2023/1674 (2023), https://eprint.iacr.org/2023/1674
25. Mosca, M., Piani, M.: 2021 quantum threat timeline report global risk institute. Global Risk Institute (2022)
26. Ngo, K.: Side-Channel Analysis of Post-Quantum Cryptographic Algorithms. Ph.D. thesis, KTH Royal Institute of Technology (2023)
27. NIST: Module-lattice-based digital signature standard. NIST FIPS (2023). https://doi.org/10.6028/NIST.FIPS.204.ipd
28. NIST: Module-lattice-based key-encapsulation mechanism standard. NIST FIPS (2023). https://doi.org/10.6028/NIST.FIPS.203.ipd
29. Oder, T., Schneider, T., Pöppelmann, T., Güneysu, T.: Practical cca2-secure and masked ring-lwe implementation. Cryptology ePrint Archive (2016). https://doi.org/10.13154/tches.v2018.i1.142-174
30. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing **2**(1), 60–66 (1973). https://doi.org/10.1137/0202007
31. Rains, E.M., Sloane, N.J.: Self-dual codes. arXiv preprint math/0208001 (2002). https://doi.org/10.48550/arXiv.math/0208001
32. Ravi, P., Roy, S.S.: Side-channel analysis of lattice-based pqc candidates. In: Round 3 Seminars, NIST Post Quantum Cryptography (2021)
33. Roy, A., Vivek, S.: Analysis and improvement of the generic higher-order masking scheme of fse 2012. Cryptology ePrint Archive, Paper 2013/345 (2013), https://eprint.iacr.org/2013/345
34. Saarinen, M.J.: Intro to side-channel security of nist pqc standards. In: PQC Seminars, NIST (2023)
35. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. pp. 124–134. Ieee (1994). https://doi.org/10.1109/SFCS.1994.365700
36. Wang, W., Méaux, P., Cassiers, G., Standaert, F.X.: Efficient and private computations with code-based masking. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 128–171 (2020). https://doi.org/10.13154/tches.v2020.i2.128-171

37. Wang, Y., Zhu, X.: A fast algorithm for the fourier transform over finite fields and its vlsi implementation. IEEE Journal on Selected Areas in Communications **6**(3), 572–577 (1988). https://doi.org/10.1109/49.1926

A Graph legend

- : Non-sensitive operation
- : Non-sensitive input/output of the algorithm
- : Non-sensitive intermediate data
- : Sensitive operation
- : Sensitive input/output of the algorithm
- : Sensitive intermediate data

B ML-KEM

B.1 PKE

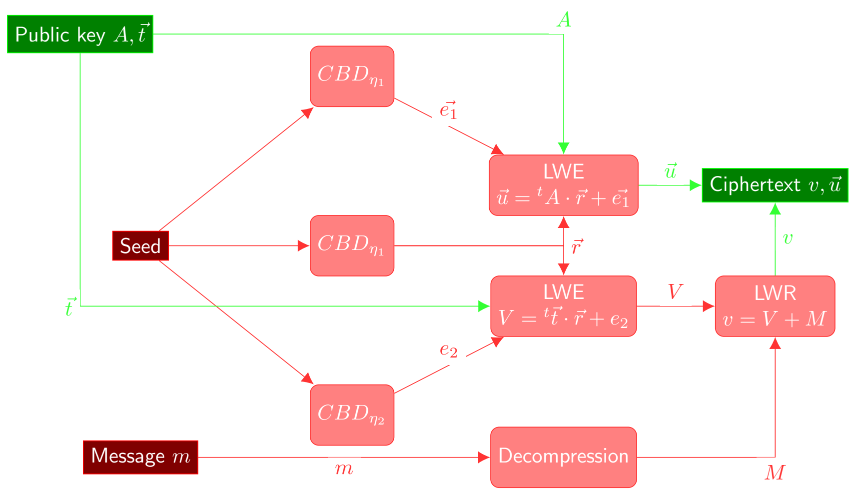


Fig. 4. Overview of the sensitive operations within PKE Encrypt

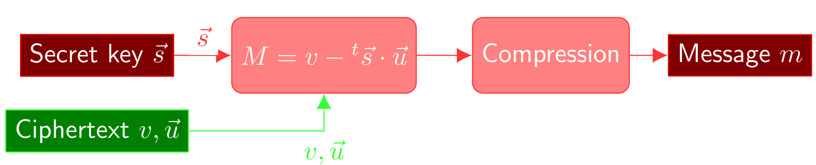


Fig. 5. Overview of the sensitive operations within PKE Decrypt

B.2 KEM

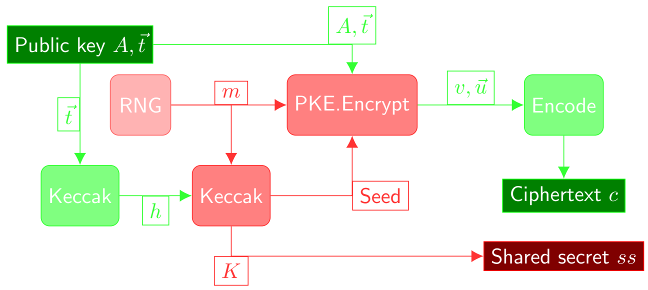


Fig. 6. Overview of the sensitive operations within the KEM Encapsulation

---

Algorithm 3 KEM Encapsulation

1: **Input:** Public key $pk = (A, \vec{t})$
2: **Output:** Ciphertext c
3: **Output:** Shared secret $K \in \{0,1\}^{256}$
4: m (256 random bits from system)
5: $m = H(m)$
6: $(K, seed) := G(m \| H(pk))$
7: $\vec{u}, v := PKE.Encrypt(pk, m, seed)$
8: $c = Encode(\vec{u}, v, d_u, d_v)$
9: **return** (c, K)

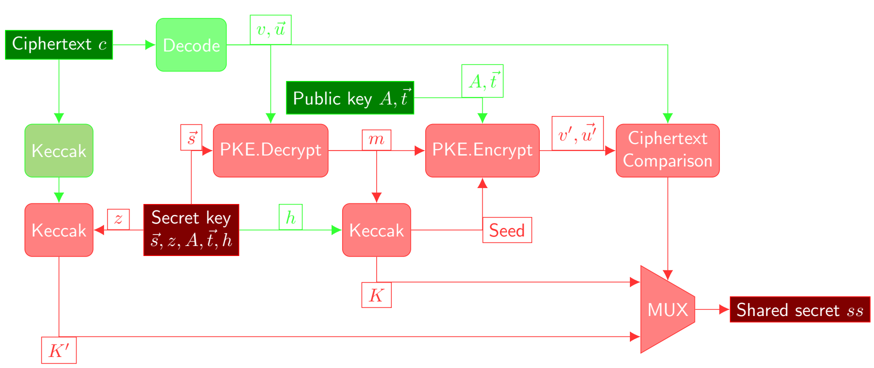


Fig. 7. Overview of the sensitive operations within the KEM Decapsulation

Algorithm 4 KEM Decapsulation

Input: Ciphertext c
Input: Secret Key $sk = (\vec{s}, pk, H(pk), z)$
Output: Shared key K
$\vec{u}, v = Decode(c, d_u, d_v)$
$m' := PKE.Decrypt(\vec{s}, \vec{u}, v)$
$(K', seed') := G(m' \| H(pk))$
$\tilde{K} = J(z \| c, 32)$ ▷ Our variation: $\tilde{K} = J(z \| H(c), 32)$
$\vec{u'}, v' := PKE.Encaps(pk, m', seed')$
$c' = (Compress_q(\vec{u'}, d_u), Compress_q(v', d_v))$
if $Compare(\vec{u'}, \vec{u}, v', v)$ **then** ▷ $Compare = 0$ if successful
    $K' = \tilde{K}$
end if
return K'

B.3 Parameters

Table 4. Parameter sets for ML-KEM

	NIST security level	n	q	sec	η_1	η_2	d_u	d_v
ML-KEM-512	I	256	3329	2	3	2	10	4
ML-KEM-768	III	256	3329	3	2	2	10	4
ML-KEM-1024	V	256	3329	4	2	2	11	5