Quasilinear Masking to Protect ML-KEM Against Both SCA and FIA*

Pierre-Augustin Berthet[1,2][0009−0005−5065−2730] (✉), Yoan Rougeolle[2][0009−0004−7088−6203], 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,yoan.rougeolle,cedric.tavernier}@hensoldt.net

Abstract. The recent technological advances in Post-Quantum Cryptography (PQC) raise the questions of robust implementations of new asymmetric cryptography 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 National Institute of Standards and Technology (NIST) as the first standard for Key Encapsulation Mechanism (KEM), taking inspiration from CRYSTALS-Kyber. We must ensure that 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). We extend previous results from a paper using Reed-Solomon codes on AES for Code-Based Masking (CBM). This work present a complete masked implementation of ML-KEM with both SCA and FIA resilience thanks to the error correcting capabilities of Code-Based Masking. Due to the structure of this masking, we propose new generic solutions to address the non-linear parts of ML-KEM, with algorithmic optimizations. To do so, we develop a new conversion methodology between boolean and arithmetic Code-Based Maskings in the specific case of ML-KEM. Performances on a laptop as well as on a SAM4S microcontroller are detailed. Security is experimentally verified by performing a Test Vector Leakage Assessment (TVLA) on a SAM4S target thanks to a Chipwhisperer Husky. We also provide formal proofs of security in the SNI model.

Keywords: Post-Quantum Cryptography · ML-KEM · Side-Channel Analysis · Masking · Code-Based Masking · Code-Based Masking Conversion

* Supported by Agence de l'Innovation de Défense, Ministère des Armées, France, Grant 2022156 Thèse CIFRE Défense.

1 Introduction

The likelihood of an efficient quantum computer in the coming 30 years is high [MP22]. Such a computer would be able to break current asymmetric cryptography primitives by taking advantage of the Shor quantum algorithm [Sho94]. The NIST launched a standardization process for PQC in 2016 [CCJ+16]. This international competition aims to standardize digital signature and KEM protocols secured against quantum and classical computers. Three signatures and one KEM were selected [AAC+22] while 4 other KEMs are heading for a fourth round to serve as alternatives in case of a cryptanalysis breakthrough [3]. At the time of writing this work, three out of four selected candidates are now in their final standard version: ML-KEM or FIPS 203 [MLK24], ML-DSA or FIPS 204 [MLD24] and SLH-DSA or FIPS 205 [SLH24].

On the other hand, we have to take into account side channel threats. Since the late 1990s and the publication of Kocher on Side-Channel Analysis [Koc96], physical attacks try to take advantage of physical leakages or faults injected within the implementation to recover sensitive data. The recent Post-Quantum Cryptography (PQC) primitives are particularly targeted as their implementation still requires secure architectures and analysis to make them robust against these physical attacks.

1.1 Background on Masking

One of the most efficient and proven countermeasure against SCA is masking [CJRR99]. The core idea is to avoid manipulating a sensitive data but instead random *shares* of it that are reassembled once the computations are done. The shares are a combination of the sensitive data and a number of random variables called *masks*. We denote by *sharing* the set of shares masking a sensitive value. The order of masking is determined by the minimum number of shares an attacker needs to recover the secret data. A high-order masking means a better security against differential attacks but it generally comes at the cost of worse performances and space.
Classical masking involves either arithmetic masking, where the random masks are subtracted or added to the secret, or boolean masking where the random masks are XORed with the secret. In this paper we use a variant of the Direct Sum Masking (DSM) introduced by Bringer et al. [BCC+14], namely a Code-Based Masking adapted from Carlet et al. [CDGT24].
We focus on ML-KEM [MLK24], inspired by CRYSTALS-Kyber [SAB+22], a post-quantum PKE/KEM. There are several publications on how to mask it with classical masking. For instance, Heinz et al. [HKL+22] proposed the first open-source implementation of a masked Kyber on microprocessor while relying on the work of Oder et al. [OSPG18] on previous lattice-based primitives. Bos

[3] SIDH/SIKE [JAC+22], one of the KEMs of the 4th round, fell victim to such breakthroughs [CD23,MMP+23,Rob23,FP22], stressing the need for alternative standards and hybridization

et al. [BGR+21] proposed a masked software implementation of Kyber while Bronchain and Cassiers [BC22] proposed new gadgets for Arithmetic to Boolean (A2B) and Boolean to Arithmetic (B2A) conversions and tested them in an open-source masked implementation of CRYSTALS-Kyber for microprocessors. When it comes to SCA and FIA resilient implementations, Heinz and Pöppelmann [HP23] proposed a combined fault and DPA protection for lattice-based cryptography. However, they only secured the linear parts of the algorithm and were not able to correct faults. Fault attacks against masked implementations of ML-KEM are a real concern, with work from Delvaux [DM21] and Kundu et al. [KCS+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 wider possibilities for 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 regarding masked ML-KEM up to our knowledge. The correction capacity can be useful for satellite applications where cosmic radiation can fault the computations.

1.2 Our Contributions

We extend results from a previous work on AES from Carlet et al. [CDGT24] to apply Code-Based Masking on a post-quantum cryptography primitive with finite fields, namely ML-KEM. We detail first- and third-order masking method for this algorithm, with 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. The only existing solution for combined protection from Heinz and Pöppelmann [HP23] only covers the Number Theoretic Transform (NTT) used in ML-KEM and does not correct errors.

By construction, Code-Based Masking is restrictive and masking some key functions requires to perform new algorithmic methods. Hence, we present in this paper some efficient alternatives to the state-of-the-art [BGR+21,BC22] regarding the masking techniques.
Noticeably, ML-KEM alternates between arithmetic computations and the use of boolean functions. We study some conversions between different Code-Based Maskings for the specific case of ML-KEM. While conversions between arithmetic and boolean masking have been widely studied, there is currently no literature on conversions between Code-Based Masking procedures.
ML-KEM contains non-linear functions like the compression function and ciphertext comparison. We propose generic masking solutions for these functions using Lagrangian interpolations. To optimize the algorithmic complexity, we use the Paterson and Stockmeyer [PS73] method to evaluate polynomials. We adapt this method to evaluate masked polynomials in a finite field of characteristic $q \neq 2$, compared to current literature only covering $q = 2$. We also use Fermat's little theorem to further reduce the cost of evaluating Lagrangian interpolations. We provide details on the complexity of evaluating such interpolations in a secure manner.

Code-Based Masking can be used to encode several sensitive data within one mask through a process introduced by Wang et al. [WMCS20] called Cost-Amortization. We study the application of this method to ML-KEM and provide some details on how to switch from a masking using Cost-Amortization to one not using it and vice versa. We perform this switch for Code-Based Masking in characteristic 2.

A Test Vector Leakage Assessment (TVLA) [GGJR$^+$11] is performed on isolated gadgets as well as on compositions of gadgets, providing experimental validation of our security claims at the first order of masking. Regarding formal security, we provide an estimation of the resilience of our design in the *SNI* model from Barthe et al. [BBD$^+$16].

We provide some performances from a *PoC* on a laptop equipped with a 11th Gen Intel(R) Core(TM) i7-11850H CPU and 16GB of RAM, as well as on a SAM4S microcontroller. The performances are compared with a state-of-the-art implementation from [BGR$^+$21].

The paper is structured as follows: in Section 2, we introduce the notations and the ML-KEM algorithm. In Section 3, we present the adaptation of the Code-Based Masking gadgets from [CDGT24] to ML-KEM. The conversion and the secure polynomial evaluation are studied respectively in Section 4 and 5. An estimation of the security based on experiments is proposed in Section 7. Finally, in Section 8, we discuss about the performances. The Section 9 concludes our paper.

2 Preliminaries

2.1 Notations

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 if an integer $\mu \neq 0 \mod q$ divides $q-1$, then we have

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

For any vector $(u_0, ..., u_{\mu-1}) \in \mathbb{F}_q^\mu$, we can associate the polynomial $U(X) = u_0 + u_1 X + \ldots + u_{\mu-1} X^{\mu-1}$. The Discrete Fourier Transform (`DFT`) and its inverse (`IDFT` or $\texttt{DFT}^{-1}$) are defined by

$$\texttt{DFT}_\omega(u_0, ..., u_{\mu-1}) = \left(U(\omega^j)\right)_{j \in [0 \cdots \mu-1]},$$

$$\texttt{IDFT}_\omega(U(1), \ldots, U(\omega^{\mu-1})) = (u_0, ..., u_{\mu-1}).$$

The $\texttt{DFT}_\omega$ operation is equivalent to a Vandermonde matrix multiplication $V(\omega)$ with $V(\omega) = (\omega^{ij})_{i,j \in [\![0,\mu-1]\!]}$ and

$$\texttt{DFT}_\omega(u_0, ..., u_{\mu-1}) = (u_0, ..., u_{\mu-1}) \times V(\omega). \tag{1}$$

We perform $\texttt{DFT}^{-1}$ by composing the `DFT` with a permutation.

Given a code length $l \in \mathbb{N}$ and a code dimension $k \in \mathbb{N}$, for length l vectors of the form $(u_0, ..., u_{k-1}, 0, \ldots, 0)$, the $\texttt{DFT}_\omega$ operation corresponds to an encoding procedure by the Reed-Solomon code denoted: RS$[l, k, l-k+1]$. A generator matrix of this code is given by the shortened matrix $(\omega^{ij})_{i \in [\![0,k-1]\!], j \in [\![0,2k-1]\!]}$. We recall some results that can be found in [RS02]: this error correcting code is classic, it is a MDS (maximum distance separable) code, which means that its minimal distance is optimal and equals $l-k+1$ where l is code length and k is its dimension. Among the good properties of these codes, we have, if R is a generator matrix of MDS code $\mathcal{R}S$ of length l and dimension k that:

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

We remind that any $[l, k, l-k+1]$-linear code can detect up to $l-k$ errors. We denote by τ the number of sensitive data encoded within one mask sharing.

Let two vectors V_1 and V_2 in $\mathbb{F}_q^l$, then we define the operation $V = V_1 \odot V_2$ with $V \in \mathbb{F}_q^l$ satisfying $\forall i \in [\![0, l-1]\!], V[i] = V_1[i] \times V_2[i]$ over $\mathbb{F}_q$.

2.2 ML-KEM

ML-KEM or FIPS 203 [MLK24] is the first post-quantum KEM standard by the NIST. It relies on several instances of the Module-LWE/LWR problems and is based on the LPR [LPR10] framework.
ML-KEM has a CPA-Secure PKE core. To ensure CCA-level of security and a KEM status, a modified version of the Fujisaki-Okamoto Transform [FO99] is used.
ML-KEM has three levels of security, with different parameter sets ([MLK24] Table 2 page 39). All sets use the same modulo, namely $q = 3329$ and message length $n = 256$. We also denote $\mathbb{F}_q[X]/(X^n+1)$ by R_q and $S_\eta := \{P \in R_q, \|P\|_\infty \leq \eta\}$ a subset of R_q.

Amongst the other notations defined by ML-KEM, we have $\lceil \cdot \rfloor$, the nearest integer with ties rounded up. It is used in the compression and decompression functions, defined as follow:

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

$$Decompress_q(\beta, d_i) = \left\lceil \frac{q}{2^{d_i}} \cdot \beta \right\rfloor, \beta \in \mathbb{F}_{2^{d_i}} \tag{3}$$

For a vector of polynomials, these two functions are applied to each coefficient of each polynomial separately.

Remark 1. For $d_i = 1$, the $Decompress_q$ function can be seen as a multiplication by a scalar, as the value β in the equation 3 can be extracted from the rounding as it can only be 0 or 1. Thus, we have $\lceil \frac{q}{2} \cdot \beta \rfloor = 1665 \cdot \beta$.

Remark 2. The compression functions are lossy:

$$\text{If } m' = Decompress_q(Compress_q(m, d_i), d_i), \text{ then } |m - m'| \leq \lceil q/2^{d_i+1} \rfloor \quad (4)$$

The compression and decompression functions are used in the message processing as well as in the ciphertext comparison. The compression is non-linear and requires a specific masking strategy. The decompression is often used on public data (apart from its usage on the message documented in Remark 1) and thus does not require to be masked.

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

$$CBD_\eta(\beta) = \sum_{i=0}^{n-1}(\sum_{j=0}^{\eta} \beta_{2i\eta+j} - \sum_{j=0}^{\eta} \beta_{2i\eta+\eta+j})X^i \text{ with } \beta \in \{0,1\}^{2\times n\times \eta} \quad (5)$$

This function is fed with a pseudo-random input β, generated with a nonce N by the PseudoRandom Function `PRF`:

$$\texttt{PRF}(seed, N) = \texttt{SHAKE256}(seed \| N). \quad (6)$$

`SHAKE256` is a hash function described in the FIPS 202 standard [Dwo15]. The counter N allows the seed reuse for the multiple values sampled during the PKE algorithms of ML-KEM. It is incremented after each call to CBD.

`KEM.Decaps` and its subfunctions `PKE.Decrypt` and `PKE.Encrypt` are presented in the Algorithm 1.

Algorithm 1: ML-KEM Decapsulation, Encrypt and Decrypt

Procedure `PKE.Encrypt`$(pk = (A, \vec{t}), seed, m)$**:**
  $\vec{r}, \vec{e_1}, e_2$ sampled with CBD from $seed$
  $\vec{u} = A \cdot \vec{r} + \vec{e_1}$
  $v = \vec{t^\perp} \cdot \vec{r} + e_2 + Decompress_q(m, 1)$
  return $c = (Compress_q(\vec{u}, d_u), Compress_q(v, d_v))$

Procedure `PKE.Decrypt`$(c = (c_u, c_v), \vec{s})$**:**
  $\vec{u} = Decompress_q(c_u, d_u), v = Decompress_q(c_v, d_v)$
  return $m = Compress_q(v - \vec{s^\perp} \cdot \vec{u}, 1)$

Procedure `KEM.Decaps`$(c = (c_u, c_v), sk = (\vec{s}, pk, h = H(pk), z))$**:**
  $m' :=$ `PKE.Decrypt`$(\vec{s}, c)$
  $(K', seed') := G(m' \| h)$
  $\tilde{K} = J(z \| c, 32)$
  $c' :=$ `PKE.Encrypt`$(pk, m', seed')$
  if $c \neq c'$ **then**
    $K' = \tilde{K}$
  end
  return K'

Remark 3. H, G and J are all different Keccak [Dwo15] instances.

More information about ML-KEM are available in the FIPS 203 standard from the NIST [MLK24] and the CRYSTALS-Kyber specification papers [SAB+22].

2.3 Formal Security Proofs and Masking

$t-$probing Security The security of the masking was first formalized by Ishai et al. in [ISW03]. They introduced the $t-$probing model. In this model an attacker has access to t wires within the circuit. These t accesses are formally represented as a physical access to the device under an attack using measurement of t probes, and it is defined as follows:

Definition 1. *(Probing security [ISW03]). A circuit is $t-$probing secure if and only if any set of at most t intermediate variables is independent from the secret.*

Compositional Security Probing security does not necessarily scale to the composition of $t-$probing secure gadgets [CPRR14]. Thus, other criterion are used in literature to ensure composition security. The most used in the literature are the $t-$Non-Interference ($t-$NI) and the $t-$Strong Non-Interference ($t-$SNI) criteria from Barthe et al. [BBD+16]. They define the notion of simulatability as follows:

Definition 2. *(Simulatability, [BBD+16, WMCS20]) Let $\mathcal{P}$ be a set of probes of a circuit (resp. gadget) $\mathtt{C}$ with input variables χ and let $\chi' \subseteq \chi$. A simulator is a randomized function $\mathtt{S}\colon \mathbb{F}_q^{|\chi'|} \to \mathbb{F}_q^{|\mathcal{P}|}$. A distinguisher is a randomized function $\mathtt{D}\colon (\mathbb{F}_q^{|\mathcal{P}|}, \mathbb{F}_q^{|\chi|}) \to \{0,1\}$. The set of probes $\mathcal{P}$ can be simulated (resp., simulated under valid input sharings) with input χ' if and only if there exists a simulator $\mathtt{S}$ such that for any distinguisher $\mathtt{D}$ and any input $x \in \mathbb{F}_q^{|\chi|}$ (resp., any valid input sharings), we have:*

$$Pr[\mathtt{D}(\mathtt{C}_{\mathcal{P}}(x), x) = 1] = Pr[\mathtt{D}(\mathtt{S}(x|_{\chi'}, \chi) = 1],$$

where the probability is over the random coins in $\mathtt{C}$,$\mathtt{S}$ and $\mathtt{D}$ and $x|_{\chi'}$ denotes the elements of x corresponding to the inputs in χ'.

There exist different strategies to prove that an implementation meets the $t-$SNI criterion such as studying the composition of $t-$NI and $t-$SNI gadgets or designing a correct circuit. Compared to the $t-$probing model, the Non-Interference model benefits from gadget composition properties. We have the following definition:

Definition 3. *(Non-Interference security [BBD+16]). A circuit is said to be $t-$Non-Interference secure ($t-$NI) if and only if any set of at most t intermediate variables can be perfectly simulated from at most t shares of each input.*

The Strong Non-Interference security is a stronger notion than Non-Interference security as it additionally guarantees the independence between input and output sharings. The latter property is very convenient to securely compose gadgets with related inputs. Hence, the composition of $t-$SNI gadgets is itself $t-$SNI. The criterion is defined as follows:

Definition 4. *(Strong Non-Interference security [BBD+16]). A circuit is said $t-$Strong Non-Interference secure ($t-$SNI) if and only if any set of at most t intermediate variables whose t_1 on the internal variables (*id est *intermediate variables except the output's ones) and t_2 on output variables can be perfectly simulated from at most t_1 shares of each input.*

A consequence, we have the following implications:

$$t - \text{SNI} \Rightarrow t - \text{NI} \Rightarrow t - \text{probing} \tag{7}$$

Regarding the composition of $t-$NI and $t-$SNI gadgets, we have the following lemma:

Lemma 1. *(Composability of $t-$NI and t-SNI gadgets [WMCS20]). A composition of gadgets is $t-$NI if all gadgets are $t-$NI or $t-$SNI, based on the following composition rule: each sharing is used at most once as input of a gadget other than $t-$SNI refresh gadget. Moreover, a composition of gadgets is $t-$SNI if it is $t-$NI and the output sharings are from $t-$SNI gadgets.*

A key gadget in the $t-$SNI model is the `Refresh` gadget. It is essential in turning $t-$NI gadgets into $t-$SNI gadgets and thus a major element within $t-$SNI model proofs. It is defined as follows:

Definition 5. *($t-$SNI Refresh gadget [BBD+16,WMCS20]). A $t-$SNI refresh gadget is a $t-$SNI gadget with one input sharing and one output sharing that ensures correctness for the identity function.*

3 Code-Based Masking for ML-KEM

Arithmetic or boolean masking can be seen through the scope of code theory as reduced parity codes. On the other hand, Direct-Sum Masking (DSM) is more generic. In this paper, we use a specific type of DSM introduced by Carlet et al. [CDGT24]: Quasilinear Code-Based Masking.

3.1 Masking gadget

The DSM method uses the following function to mask an element x:

$$\forall \vec{x} \in \mathbb{K}, \vec{r} \text{ random} \in \mathbb{K}, \texttt{Mask}_{DSM}(\vec{x}, \vec{r}) = \vec{x} \cdot G + \vec{r} \cdot H \tag{8}$$

In Equation 8, the matrix G refers to a generator of a code $\mathcal{C}$ and H to a parity-check matrix for the code $\mathcal{C}$.

Carlet et al. [CDGT24] use the Reed-Solomon codes with the DSM framework. This allows the use of the Discrete Fourier Transform (DFT) in many operations within this Code-Based Masking, like fault detection, masked multiplication and all of this operation are performed in quasilinear time. The `Mask` function is defined as follows:

$$\texttt{Mask}(x) = (x, \vec{r}) \cdot A^{-1} \cdot V(\omega) \tag{9}$$

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

The choice of α and ω depends on the code's field $\mathbb{F}_q$. Given field order q, (q_1, q_2) coprime subfactors of $q-1$ and a primitive root ν in $\mathbb{F}_q$, we take $\alpha = \nu^{q_2}$ and $\omega = \nu^{q_1}$. These conditions are sufficient according to [CDGT24]. The product $A^{-1} \cdot V(\omega)$ can be seen as an encoder tied to a Reed-Solomon code.

Remark 4. The matrice $V(\omega)$ is a Vandermonde matrice. By setting $(x, \vec{r}) = (c_0, c_1, \dots, c_{k-1}) = \vec{c}$, $\vec{c} \times A^{-1} = (p_0, p_1, \dots, p_{k-1})$ and $P(X) = \sum_{i=0}^{k-1} p_i X^i$, another equation for this Code-Based Masking is $\texttt{Mask}(x) = \texttt{DFT}_\omega(P(X))$. Interestingly, by setting $(u_j)^i = A_{i,j}$, we have that P satisfies $P(u_i) = c_i$.

Formal security of the masking gadget We first introduce the notion of Generic encoder from [WMCS20]:

Definition 6. *(Generic encoder, [WMCS20]). Let τ, k and l be three positive integers, a generic encoder is a code-based encoder with the following restriction on $R \in \mathbb{F}_q^{(\tau+m)\times l}$:*

- *G is the $\tau \times l$ upper part of R, which is the part multiplied by the encoded variables. $\texttt{Rank}(G) = \tau$ and we refer to $\mathcal{C}_G$ for the code generated by G.*
- *H is the $(k-\tau) \times l$ lower part of R, which corresponds to the part multiplied by the randomness. $\texttt{Rank}(H) = (k-\tau)$ and we refer to $\mathcal{C}_H$ for the code generated by H.*
- *$\mathcal{C}_G \cap \mathcal{C}_H = \{0\}$.*

The encoding procedure corresponds to the operation

$$\texttt{Mask} : x \in \mathbb{F}_q^\tau \to (x, r) \in \mathbb{F}_q^k \to (x, r) \times R.$$

Proposition 1. *(Generic encoders and $t-$privacy, [WMCS20]) . Let `Mask` be a generic encoder, let d denote the minimal distance of $\mathcal{C}_{H^\perp}$, then `Mask` is $t-$private with:*

$$t = \max\{i \in \{0, \dots, l\} : \forall w \in \{\mathcal{C}_{H^\perp}\}_i, Gw^T = 0^T\}, \tag{10}$$

$$= d - 1 + \max\{i \in \{0, \dots, l-d+1\} : \forall w \in \{\mathcal{C}_{H^\perp}\}_{d-1+i}, Gw^T = 0^T\}. \tag{11}$$

Proposition 2. `Mask` *is a $(d-\tau)-$private generic encoder.*

Proof. `Mask` has been proven $(d-\tau)-$probing secure by [CDGT24] and by construction it satisfies the definition 6 and the proposition 1. □

3.2 Addition and scaling

Let $\vec{z} = \texttt{Mask}(x)$, $\vec{z'} = \texttt{Mask}(x')$ and $\lambda \in \mathbb{F}_q$. Due to the linear nature of the Reed-Solomon codes, we can define the addition and scaling as follows:

$$\texttt{AddMask}(x, x') = \texttt{Mask}(x + x') = \vec{z} + \vec{z'} \tag{12}$$

$$\texttt{ScalMask}(\lambda, x) = \texttt{Mask}(\lambda \cdot x) = \lambda \cdot \vec{z} \tag{13}$$

Formal security of the addition and scaling We use the Theorem 3 from [WMCS20], stating the following:

Theorem 1. *([WMCS20])Let $x, y \in \mathbb{F}_q$,* `Mask` *the $(d-\tau)-$private generic encoder with $R = A^{-1}V(\omega) \in \mathbb{F}_q^{k \times l}$. Then, the gadget* `AddMask` *with inputs $x' =$ $\texttt{Mask}(x)$, $y' = \texttt{Mask}(y)$ and output z' ensures that $\texttt{Unmask}(z') = x + y$, and* `AddMask` *is $t-$NI for any positive integer t such that $t \leq (d - \tau)$.*

Proof. [WMCS20]). By construction, we have $\texttt{Unmask}(z') = \texttt{Unmask}(x' + y') = \texttt{Unmask}(\texttt{Mask}(x) + \texttt{Mask}(y)) = x + y$ and we get the correctness. Since the operations are performed share-wisely, any probe (internal and output) in `AddMask` has one of the following three forms: $x'[i], y'[i]$ or $x'[i] + y'[i]$ where $i \in [1..l]$. Let I be the indexes appearing in the t probes, by construction $|I| \leq t$, and $x'[I]$ and $y'[I]$ are sufficient to simulate all the probes. Therefore, for any set $\mathcal{P}$ of $t \leq l$ probes we can build a simulator S such that: $Pr[D(C_{\mathcal{P}}(x', y'), x', y') = 1] = Pr[D(S(x'[I], y'[I]), x', y') = 1]$,for any distinguisher D.

Theorem 2. *The gadget* `ScalMask` *is $t-$NI secure for any positive integer t such that $t \leq (d - \tau)$.*

Proof. This gadget performs a linear operation on each share independently. Similarly to the proof of Theorem 1, it is $t-$NI secure. □

3.3 Refresh gadget

The `Refresh` gadget is performed by adding to the masked value an encoding of 0:

$$\texttt{Refresh}(\texttt{Mask}(x)) = \texttt{Mask}(x) + \texttt{Mask}(0) \tag{14}$$

Theorem 3. *The refresh gadget* `Refresh` *(Equation 14) is $(d-\tau)-$SNI secure.*

Proof. Our `Refresh` uses the same construction as the one from [WMCS20] and is consequently $(d - \tau)-$SNI as well. □

3.4 Multiplication

We have in this case a difference of sign with [CDGT24], thus we summarize the computation below. If $\vec{z} = \texttt{Mask}(x)$ and $\vec{z}\,' = \texttt{Mask}(y)$, then

$$\vec{z} \odot \vec{z}\,' = \texttt{DFT}_\omega(a_0, \ldots, a_{k-1}, 0, \ldots, 0) \odot \texttt{DFT}_\omega(a'_0, \ldots, a'_{k-1}, 0, \ldots, 0). \tag{15}$$

The polynomial obtained by performing $\texttt{DFT}_\omega^{-1}(\texttt{DFT}_\omega(P_x)\times\texttt{DFT}_\omega(P_y)) = P_x(X)\times P_y(X) = C(X)$ is a $2k-2$ degree polynomial, which satisfies $C(u_i) = P_x(u_i)\times P_y(u_i) = x_i y_i$ for i in $\{0,\dots,\tau-1\}$. Now we build a degree $k-1$ polynomial $D(X)$ that satisfies $D(u_i) = C(u_i)$ for all $0 \le i \le \tau-1$. We introduce the polynomials

$$U_j(X) = u_j^{k-1}\frac{(X-u_0)\cdots(X-u_{j-1})(X-u_{j+1})\cdots(X-u_{\tau-1})}{(u_j-u_0)\cdots(u_j-u_{j-1})(u_j-u_{j+1})\cdots(u_j-u_{\tau-1})}, \tag{16}$$

which satisfy $U_j(u_j) = u_j^{k-1}, U_j(u_i) = 0\ \forall\, i \in \{0,\dots,\tau-1\}\backslash\{j\}$ and $\deg(U_j(X)) = \tau-1$. We then build

$$\begin{aligned} D(X) = c_0 + c_1X + \cdots + c_{k-1}X^{k-1} + (c_kX + \cdots + c_{2k-\tau-1}X^{k-\tau})\sum_{j=1}^{\tau} U_j(X) \\ + \sum_{i=1}^{\tau-1} c_{2k-\tau-1+i}\sum_{j=1}^{\tau} U_j(X)u_j^{k-\tau+i}, \end{aligned}$$

which satisfies $D(u_i) = C(u_i) = x_iy_i$ of $i \in \{0,\dots,\tau-1\}$. Hence, we get, with the difference with [CDGT24] highlighted in red:

$$\begin{aligned} \texttt{DFT}_\omega(D(X)) = \quad & \texttt{DFT}_\omega(C(X)) \\ & - \texttt{DFT}_\omega(c_kX^k + \cdots + c_{2k-2}X^{2k-2}) \\ & + \texttt{DFT}_\omega(c_kX + \cdots + c_{2k-\tau-1}X^{k-\tau}) \odot \vec{u} \\ & + \sum_{i=1}^{\tau-1} c_{2k-\tau-1+i}\cdot G_i = \texttt{Mask}(x\odot y), \end{aligned}$$

where $G_i = \texttt{DFT}_\omega(\sum_{j=1}^{\tau} U_j(X)u_j^{k-\tau+i})$ for $i \in \{1,\dots,\tau-1\}$ as well as $\vec{u} = \texttt{DFT}_\omega(\sum_{j=1}^{\tau} U_j(X))$ are precomputed values, and $c_k,\dots,c_{2k-2}$ is given by the function $\texttt{extractLastCoefficients}(\vec{z}\odot\vec{z}\,')$. $\texttt{extractLastCoefficients}$ has been defined in [CDGT24], that is to say $\texttt{IDFT}_\omega(\vec{z}\odot\vec{z}\,') = (c_i)_{i\in\{0,\dots,l-1\}} = C(X)$, then if we denote $Z = \vec{z}\odot\vec{z}\,'$, by definition $c_{j+k} = \sum_{i=0}^{l-1} Z_i\omega^{-i(j+k)} = \sum_{i=0}^{n-1}(Z_i\omega^{-ik})\omega^{-ij}\ \forall\ 0 \le j \le k-1$ and $(c_{j+k})_{j\in\{0,\dots,k-1\}}$ is obtained from $\texttt{IDFT}\left((Z_i\omega^{-ik})_{0\le i\le l-1}\right)$.

Finally, if we denote $\phi(C,\omega) = -\texttt{DFT}_\omega(c_{k+1}X^k+\cdots+c_{2k-2}X^{2k-2})+\texttt{DFT}_\omega(c_kX+\cdots+c_{2k-\tau-1}X^{k-\tau})\odot\vec{u}+\sum_{i=1}^{\tau-1} c_{2k-\tau-1+i}\cdot G_i$ where C represents the $k-1$ last coefficients of $\texttt{IDFT}(\texttt{Mask}(x)\odot\texttt{Mask}(y))$, then we get that

$$\texttt{Mask}(x\odot y) = \texttt{Mask}(x)\odot\texttt{Mask}(y) + \phi(C,\omega). \tag{17}$$

Formal security and fault resilience of the multiplication First, we detail the tree decomposition for the $\texttt{DFT}$ in Figure 1. We adopt this peculiar structure to ensure Theorem 4. In the Figure 1, we use the following notations: for readability, we omit the (X) in $P(X)$, also we use brackets to express the modulo

$P[Q] \equiv P(X)$ modulo $Q(X)$. The Q_i and T_i polynomials are moduli polynomials for the DFT in accordance with the construction from Wang and Zhu [WZ88]. The R_i polynomials are degree i randomized polynomials. All three instances of R_i within the tree are randomized and different.

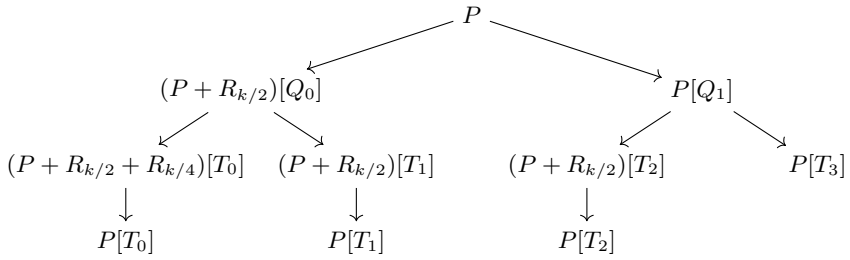


Fig. 1: Tree for the DFT

Theorem 4. *The* DFT *described in Figure 1 is* $(d-\tau)-$*SNI secure.*

Proof. The tree in Figure 1 has a recursive structure. Thus, we demonstrate correctness and $(d-\tau)-$SNI security for one branching and, thanks to the composition Lemma 1, the entire tree will be $(d-\tau)-$SNI secure.
We remind some properties around Chinese Remainder Theorem: Let Q_0 and Q_1, such that $\gcd(Q_0, Q_1) = 1$, the the following morphism is an isomorphism:

$$\varphi : \frac{\mathbb{Z}[X]}{Q_0 Q_1} \mapsto \frac{\mathbb{Z}[X]}{Q_0} \times \frac{\mathbb{Z}[X]}{Q_1}$$

$$P \quad \to (P[Q_0], P[Q_1]).$$

It means in terms of linear algebra that there exists two matrices A and B such that

$$\varphi(p_0, p_1, \ldots, p_d) = (p_0, p_1, \ldots, p_d) \times (A, B)$$

By construction, the DFT coming from [WZ88] is such that at any node, $\gcd(Q_0, Q_1) = 1$. As a consequence, as vector spaces, we have $A \cap B = \{0\}$.
Due to the linear bijection, simulating a set of I input requires a total of at least I (internal and output) probes, then the circuit of φ is $t-$NI $\forall t \leq (d-\tau)$.

If the circuit computing $\varphi(P)$ is $t-$NI, then it is necessary to refresh one of the leaf: let's chose $P_1 = P \mod Q_1 + R$ where R is a random degree $k/2$ polynomial such that $R(\alpha) = 0$. Then the Chinese Remainder Theorem states that there exists some polynomials u and v such that $uQ_0 + vQ_1 = 1$.

We remark that $P' = P + uQ_0R$ satisfies $\varphi(P') = (P_0, P_1)$ and $P'(\alpha) = P(\alpha)$, thus the scheme including the refresh is correct and $(d-\tau)-$SNI secure. Hence, the entire tree is by recursion correct and $(d-\tau)-$SNI secure. □

In order the retain the error-correcting capability of the masking throughout the multiplication computation, we take inspiration from the work of Berndt et al. [BEF+23]. The multiplication is described in Figure 2. To compute $\texttt{DFT}(P(X)P'(X))$, we decompose $P(X) = P_0(X) + X^{k/2}P_1(X) = \texttt{DFT}^{-1}(\vec{z})$. We do the same for $\vec{z'}$ to compute $P'_0(X)$ and $P'_1(X)$. Once these four polynomials are extracted, we compute

$$P(X)P'(X) = P_0(X)P'_0(X) + X^{k/2}(P_0(X)P'_1(X) + P'_0(X)P_1(X)) \\ +X^k(P_1(X)P'_1(X)).$$

We do so in the DFT domain by computing each $P_iP'_j$ separately and multiplying the result by the proper evaluation of X^k or $X^{k/2}$. This is denoted by the Eval function. We reassemble each result by first performing a Refresh and then summing them up together to obtain the final result.

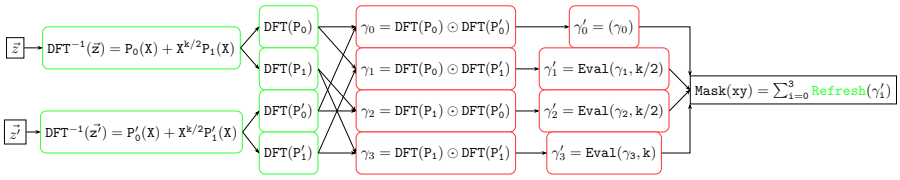


Fig. 2: Multiplication gadget with a NI and SNI color code

Theorem 5. *The multiplication gadget described in Figure 2 is $(d-\tau)-$SNI secure.*

Proof. In this proof we use the composition Lemma 1. We justify the color coding for each block, with green blocks or functions being $(d-\tau)-$SNI and red ones being $t-$NI.
The inverse DFT is, due to the $(d-\tau)-$SNI security of the DFT proved in Theorem 4, $(d-\tau)-$SNI as well as we simply perform a composition by a permutation matrix. The permutation is $t-$NI as it simply switches the shares but the composition with a $t-$SNI gadget turns it into a $t-$SNI gadget as well. Extracting the P_i and P_j can be done with independent wiring within the circuit and thus is not considered in our formal proof as it does not impact the security.
Multiplying coordinates-wise the DFTs is a $t-$NI secure computation as it is performed on each share separately.
Similarly, the Eval function is $t-$NI secure as it is a multiplication by a precomputed scalar and thus a linear operation performed on each share separately.
Finally, it is important to perform a Refresh before adding each computation together. We proved in Theorem 3 that our Refresh gadget is $(d-\tau)-$SNI. Thus, by applying the composition lemma, as the wires on which we compute γ_i and γ'_i

independently all end up by a $(d-\tau)-$SNI secure gadget, they form separately $(d-\tau)-$SNI secure gadgets. Even if the final addition is a $t-$NI secure gadget, thanks once again to the composition lemma, we have that our multiplication is a $(d-\tau)-$SNI secure gadget. □

More details are available in the original Quasilinear Code-Based Masking paper from Carlet et al. [CDGT24].

3.5 Codes For ML-KEM

We list the possible Reed-Solomon encoders for several masking order and each characteristic for ML-KEM in Table 1. We choose the length of the encoders to ensure we can easily perform the `DFT`. This length must then be a factor of $q-1$.

Order	1	2	3	4	5	6
Over $\mathbb{F}_q$	$\mathcal{RS}[4,2,3]$	$\mathcal{RS}[8,3,6]$	$\mathcal{RS}[8,4,5]$	$\mathcal{RS}[13,5,9]$	$\mathcal{RS}[13,6,8]$	$\mathcal{RS}[13,7,7]$
Over $\mathbb{F}_{16}$	$\mathcal{RS}[3,2,2]$	$\mathcal{RS}[5,3,3]$	None	None	None	None
Over $\mathbb{F}_{64}$	$\mathcal{RS}[3,2,2]$	$\mathcal{RS}[7,3,5]$	$\mathcal{RS}[7,4,4]$	$\mathcal{RS}[9,5,5]$	None	None
Over $\mathbb{F}_{256}$	$\mathcal{RS}[3,2,2]$	$\mathcal{RS}[5,3,3]$	$\mathcal{RS}[15,4,12]$	$\mathcal{RS}[15,5,11]$	$\mathcal{RS}[15,6,10]$	$\mathcal{RS}[15,7,9]$

Table 1: Possible encoders for different fields and masking order up to order 6

In this paper we only present performances for orders 1 and 3. We choose to skip order 2 as the encoder over $\mathbb{F}_q$ for order 2 has the same length as the one for order 3, thus resulting in similar performances over $\mathbb{F}_q$. However, it forces us to use an encoder over $\mathbb{F}_{64}$ instead of $\mathbb{F}_{16}$ for boolean functions masked at the third order. As most of the computations are performed over $\mathbb{F}_q$, the drawback of using a bigger field in characteristic 2 is compensated by having a higher order security.

Remark 5. We use Code-Based Masking over $\mathbb{F}_q$ to mask each polynomial coefficient in each vector or matrix object within ML-KEM. However, contrary to [CDGT24] where entire bytes are masked, we only mask one bit per code word when using Code-Based Masking over $\mathbb{F}_{16}$ or $\mathbb{F}_{64}$. This is due to the operations within Keccak, which require to manipulate bits separately.

We give the first order masking matrices for both fields as example. For $\mathbb{F}_{16} = \mathbb{F}_2[X]/ < X^4 + X + 1 >$, ν is the generator, $\alpha = \nu^3, \omega = \nu^5$,

$$A^{-1} \cdot V(\omega) = \begin{pmatrix} 14 & 10 & 6 \\ 15 & 7 & 5 \end{pmatrix}, \tag{18}$$

For $\mathbb{F}_q$, we take $\nu = 17, \omega = \nu^{\frac{q-1}{4}} = 1729, \alpha = \nu^{\frac{q-1}{13}} = 2970$,

$$A^{-1} \cdot V(\omega) = \begin{pmatrix} 103 & 2590 & 1545 & 2387 \\ 3227 & 740 & 1785 & 943 \end{pmatrix}. \tag{19}$$

In this work we will only focus on the non-linear part of ML-KEM. As we use a linear code, then any linear function within ML-KEM can be easily masked.

4 Conversion

By construction, some data in ML-KEM are first processed in $\mathbb{F}_q$ and then used with boolean operands. To ensure that the computations remain secure, the state-of-the-art of the masked implementations of ML-KEM [BGR+21,BC22] uses to switch between arithmetic and boolean masking considering some conversions. While conversions between arithmetic and boolean masking and back are well documented in the literature, there is no prior work to this paper on the matter of masking conversions in the case of Code-Based Masking up to our knowledge. In this section we detail how to perform such conversions in the specific case of ML-KEM.

4.1 Overall Idea

Performing Arithmetic to Boolean conversion can be interpreted as a conversion from a field of characteristic other than 2 to a field of characteristic 2. Both representations must be equipped of masking methods with similar probing orders. We have seen previously that $\mathbb{F}_q$ can be equipped with Reed-Solomon Code-Based Masking. According to the parameters, we can benefit of the Fast Fourier calculation or at worst, simply a Vandermonde matrix multiplication. The results of [CDGT24] state that we have such an encoder over characteristic 2 fields and we consider the characteristic 2 field $\mathbb{F}_{2^b}$ with $b \in \mathbb{N}^*$.

There is no common subfield between $\mathbb{F}_{2^b}$ and $\mathbb{F}_q$. However it is possible to build $(\mathbb{F}_2, \otimes, \oplus)$ as a subset of $(\mathbb{F}_q, \times, +)$. We have the following properties,

$$\forall (x,y) \in \{0_q, 1_q\}^2, \begin{cases} x \times y \Leftrightarrow x \otimes y; \\ x + y - 2 \times x \times y \Leftrightarrow x \oplus y. \end{cases} \tag{20}$$

We also have an inverse relation:

$$\forall (x,y) \in \{0_{2^b}, 1_{2^b}\}^2, \begin{cases} x \otimes y \Leftrightarrow x \times y; \\ x \oplus y \text{ with the carry } x \otimes y \Leftrightarrow x + y. \end{cases} \tag{21}$$

We deduce that, given $(a,b) \in \{0_q, 1_q\}^2$, $\texttt{Mask}_q(a \oplus b) = \texttt{Mask}_q(a) + \texttt{Mask}_q(b) - 2 \cdot \texttt{Mask}_q(a) \star \texttt{Mask}_q(b)$ and $\texttt{Mask}_q(a \otimes b) = \texttt{Mask}_q(a) \star \texttt{Mask}_q(b)$.

Remark 6. If the value masked is not certain to be either 0 or 1, it is necessary to obtain its binary decomposition before performing the conversion. This can be performed securely with Lagrange's interpolations. This is however costly, with an estimated complexity of $\lceil log(q) \rceil \times (2\lfloor \sqrt{q} \rfloor - 1)$ masked multiplications in the worst case[4]. As a result, $\lceil q \rceil$ conversions must be performed.

[4] Some interpolations might have a structure and thus the possibility of reducing the degree of the evaluated polynomial through variable changes.

The overall strategy consists in interpolating the masking operations from one field with the operations of the other field. This allows to perform the $\texttt{Mask}_2$ and $\texttt{Unmask}_2$ operations in $\mathbb{F}_q$ and the $\texttt{Mask}_q$ and $\texttt{Unmask}_q$ operations in $\mathbb{F}_{2^b}$.

4.2 From $\mathbb{F}_q$ to $\mathbb{F}_{2^b}$

We want to convert a length $l_2 \in \mathbb{N}$ Reed-Solomon encoder defined over $\mathbb{F}_q$ in an encoder defined over $\mathbb{F}_{2^b}$ that we denote M_2 and we must evaluate

$$\texttt{Mask}_2(x) = ((x, \vec{r}) \otimes M_2) = \left((0, \vec{r}) \otimes M_2 \oplus (x, \vec{0}) \otimes M_2\right). \tag{22}$$

The first step consists in computing $R = (0, \vec{r}) \otimes M_2$ and we mask 0:

$$\vec{F} = (f_0, f_1, \ldots, f_{l_2-1}) = \texttt{Mask}_2(0). \tag{23}$$

Then, we mask every bits of each coordinates of $\vec{F}$ with $\texttt{Mask}_q$:

$$R = \begin{pmatrix} \texttt{Mask}_q(f_0^{(0)}) & \ldots & \texttt{Mask}_q(f_{l_2-1}^{(0)}) \\ \ldots & \ldots & \ldots \\ \texttt{Mask}_q(f_0^{(b-1)}) & \ldots & \texttt{Mask}_q(f_{l_2-1}^{(b-1)}) \end{pmatrix} \tag{24}$$

We compute now $L = (x, \vec{0}) \otimes M_2$ and we only require the first line of the encoder that we denote $(c_0, \ldots, c_{l_2-1})$. Hence, we performe:

$$L = \begin{pmatrix} c_0^{(0)} \cdot \texttt{Mask}_q(x) & \ldots & c_{l_2-1}^{(0)} \cdot \texttt{Mask}_q(x) \\ \ldots & \ldots & \ldots \\ c_0^{(b-1)} \cdot \texttt{Mask}_q(x) & \ldots & c_{l_2-1}^{(b-1)} \cdot \texttt{Mask}_q(x) \end{pmatrix} \tag{25}$$

Finally, the conversion is performed by XORing L and R using the interpolation $S = L \oplus R = L + R - 2LR$. The final step remains to apply $\texttt{Unmask}_q$ with $\texttt{Unmask}_q(S) = \texttt{Mask}_2(x)$.

4.3 From $\mathbb{F}_{2^b}$ to $\mathbb{F}_q$

The strategy is similar to the previous conversion except the final steps where we must perform $S = L + R$ by interpolating the operand "+" as the binary addition "$+_2$" columns per columns. Hence, by denoting $\vec{m} = \texttt{Mask}_2(x)$, $G = (g_0, \ldots, g_{l_q-1}) = \texttt{Mask}_q(0)$ and $(h_0, \ldots, h_{l_q-1})$ the first line of the length l_q Reed-Solomon encoder over $\mathbb{F}_q$, then we have:

$$S = \begin{pmatrix} \texttt{Mask}_2(g_0^{(0)}) & \ldots & \texttt{Mask}_2(g_{l_q-1}^{(0)}) \\ \ldots & \ldots & \ldots \\ \texttt{Mask}_2(g_{l_q-1}^{(11)}) & \ldots & \texttt{Mask}_2(g_{l_q-1}^{(11)}) \end{pmatrix} +_2 \begin{pmatrix} h_0^{(0)} \otimes \vec{m} & \ldots & h_{l_q-1}^{(0)} \otimes \vec{m} \\ \ldots & \ldots & \ldots \\ h_0^{11} \otimes \vec{m} & \ldots & h_{l_q-1}^{(11)} \otimes \vec{m} \end{pmatrix}, \tag{26}$$

and the final step consist in computing $\texttt{Unmask}_2(S)$ to recover $\texttt{Mask}_q(x)$.

4.4 Conversions in ML-KEM

The state-of-the-art of the implementations [BGR+21,BC22] relies heavily on conversions to perform the non-linear functions of ML-KEM whereas our conversions can be performed only when we have the guarantee that the masked values are binary values. Thus, we cannot rely on their designs for our masking and we must propose some new methodologies to mask the non-linear functions of ML-KEM. In particular we remark that the message compression is a function from `PKE.Decrypt` defined over $\mathbb{F}_q$ that outputs 256 elements from $\mathbb{F}_q$ which are "0" or "1". These outputs are interpreted as bits over $\mathbb{F}_2$ and are processed by a Keccak instance to generate a seed bitstring.

The seed bistring serves as an input to the CBD instances in `PKE.Encrypt` during `KEM.Decaps`. Within these instances, bits are summed and subtracted according to Equation 5 in order to compute a random bounded value. Thus, ML-KEM requires to convert from $\mathbb{F}_q$ to $\mathbb{F}_{2^b}$ after the compression in `PKE.Decrypt` and to perform the inverse conversion during the CBD instances.

Furthermore, to reduce the number of conversions performed, we propose to slightly modify Equation 5 by only adding bits together and subtracting the resulting bias afterwards:

$$CBD_\eta(\beta) = \sum_{i=0}^{n-1}(\sum_{j=0}^{\eta}(\beta_{2i\eta+j} + \beta_{2i\eta+\eta+j}) - \eta)X^i \text{ with } \beta \in \{0,1\}^{2\times n\times\eta} \quad (27)$$

Operations in blue are performed with a binary addition. We can compute the binary decomposition of the sum from the bits directly by using the boolean functions. We then convert these results to Code-Based Masking in $\mathbb{F}_q$, add the elements of the decomposition together and finally subtract the bias. The impact on the number of conversions required is highlighted in Table 2:

Table 2: Number of conversions to perform in different ML-KEM-512/-768/-1024 functions for various strategies

Strategy	Key Generation	KEM Encaps or Decaps
Immediate Conversion	6144/6144/8192	7168/7168/9216
Delayed Conversion	3072/4608/6144	3840/5376/6912

5 Message Compression and Ciphertext Comparison

The state-of-the-art of ML-KEM masked implementations [BC22,BGR+21] proposes specific solutions to mask the non-linear parts of ML-KEM by using boolean formulae, where conversions are largely involved. To develop alternatives to their methodologies, we present in this section generic solutions to mask the message compression and the ciphertext comparison within ML-KEM which can be applied to any type of masking.

Paterson-Stockmeyer Fast Polynomial Evaluation in Characteristic $q \neq 2$ In ML-KEM, both the message compression and the ciphertext comparisons acts as "mappings" of values from $\mathbb{F}_q$ to $\{0,1\}$. Mathematically, this is equivalent to consider Lagrangian interpolations. Consequently, our methodology to securely compute these functions relies on evaluating polynomials. We use the work from Paterson and Stockmeyer [PS73] to minimize the number of sensitive multiplications required to evaluate these polynomials. Sensitive multiplications are more expensive than additions or scalar multiplications, thus we manage to reduce their number.

Remark 7. The adaptation of Paterson and Stockmeyer [PS73] for masking in characteristic 2 has been covered by Roy and Vivek [RV13].

Theorem 6. *([PS73]) Let P be a public polynomial and deg its degree. Let's assume first that $s = \sqrt{deg}$ is an integer for simplicity. We notice that $P(X)$, where X is the sensitive data we mask, can be written as:*

$$\begin{aligned} P(X) = P_0(X) + X^s P_1(X) + \ldots + X^{deg-s} P_{s-1}(X), \\ \text{with } \forall i \in [\![0, s-2]\!], \deg(P_i(X)) \leq s-1 \text{ and } \deg(P_{s-1}(X)) = s \end{aligned} \tag{28}$$

Then evaluating this polynomial in X can be done in $\mathcal{O}(2s)$ sensitive multiplications.

5.1 Message Compression

In the state-of-the-art of masked implementations [BGR+21,BC22], the masked compression is performed using A2B conversions and boolean formulae. As stated in Section 4, A2B conversions using Code-Based Masking are costly. Thus, we use an alternative methodology to perform the message compression by proposing and evaluating a variant of Lagrange interpolations.

We see $Compress_q(M = v - \vec{s}^T \cdot \vec{u}, 1)$ in `PKE.Decrypt` as a mapping of elements from $\mathbb{F}_q$ to either 0 or 1. If we perform a Lagrange interpolation, we have to evaluate a degree 3330 polynomial. However, we choose to only evaluate the part of the Lagrange interpolation mapping points to 0. This results in a polynomial P of degree 1665. By factoring the $(X-0)$ polynomial from P and using the symmetric nature of the set of points mapped to 0, we evaluate a degree 832 polynomial and then multiply by the evaluation of X:

$$P(X) = X \cdot \prod_{i=1}^{832} (X^2 - i^2). \tag{29}$$

Evaluating Equation 29 performs the mapping to 0. However, the results which should be mapped to 1 are mapped to random non-zero values in $\mathbb{F}_q$. To map to 1, instead of using the normal Lagrange interpolation, we use Fermat's little theorem which stipulates that, if q is prime, then $\forall \beta \in \mathbb{Z}$ such as $q \nmid \beta, \beta^{q-1} \equiv 1 \bmod q$. Thus, we use the methodology described Algorithm 2:

Algorithm 2: Fast Polynomial Evaluation Methodology

Input: A value $\beta \in \mathbb{F}_q$;
Input: A function F mapping $\mathbb{F}_q$ to $\{0,1\}$;
Input: A polynomial $P_F \in \mathbb{F}_q[X]$ performing the F mapping to 0;
Output: $F(\beta) \in \{0,1\}$;
Procedure:
  $\gamma = P_F(\beta)$;
  return γ^{q-1} *mod* q

Remark 8. The cost of Algorithm 2 in terms of sensitive multiplications is equal to the cost of evaluating P_F added to the cost of a fast exponentiation algorithm. In ML-KEM case, we perform 13 sensitive multiplications after the evaluation of P_F.

We will denote the mapping done by the message compression by FM and thus the associated polynomial by P_{FM}.

5.2 Ciphertext Comparison

In the reference implementation [MLK24], the ciphertext comparison is performed by compressing newly generated ciphertexts $\vec{u'}$ and v' and ensuring that there is a strict equality with the received ciphertext c. While we propose in Algorithm 2 an efficient method to perform the message compression, the mappings induced by $Compress_q(\cdot, d_u)$ and $Compress_q(\cdot, d_v)$ do not map towards either 0 or 1. However, the output of the ciphertext comparison does.

To obtain a mapping towards $\{0,1\}$ from polynomial coefficients of the ciphertexts, we use a similar strategy to Bos et al. [BGR+21] masked ML-KEM. Instead of compressing the newly generated ciphertexts, we compare the decompression of the received ciphertext with the newly generated ones. However, as stipulated in Remark 2, there is a loss of information. Thus, instead of checking for strict equality, we ensure that the biases between the received ciphertext in its decompressed state and the newly generated ones do not exceed some specific values tied to the parameters d_u and d_v. This is equivalent to check that $\|\vec{u'} - \vec{u}\|_\infty \leq 2$ and $\|v' - v\|_\infty \leq 104$ in the case of ML-KEM-512.

A first step is thus to perform $\vec{u'} - \vec{u}$ and $v' - v$. As the ciphertexts comparison should map to either 0 or 1, we propose to check the inequalities for each polynomial coefficient separately by mapping them to either 0 or 1 and then perform a masked arithmetized OR[5] of every results. This OR is arithmetized to avoid using costly conversions.
To check $\|U = (\vec{u'} - \vec{u})\|_\infty \leq 2$, we use a mapping denoted by FU such as, if $U \in [\![-2, 2]\!]$, $FU(U) = 0$, else $FU(U) = 1$. The associated polynomial mapping to 0 denoted by P_{FU} is $P_{FU}(X) = X(X^2 - 1)(X^2 - 4)$ which only requires 3

[5] $\forall a, b \in \{0,1\}^2, a\ OR\ b = a + b - a * b$

sensitive multiplications. We use Algorithm 2 to compute the mapping to 1.
To check $\|V = (v' - v)\|_\infty \leq 104$, we use a mapping denoted by FV such as, if $V \in [\![-104, 104]\!]$, $FV(V) = 0$, else $FV(V) = 1$. We will denote the associated polynomial by P_{FV}. Due to the symmetric nature of the set of points mapped to 0 by FV, the degree of P_{FV} can be reduced similarly to FM and we evaluate a polynomial of degree 104 then multiply by the evaluation of X. Mapping to 1 is once again performed according to Algorithm 2.

5.3 Complexity

We use the methodology proposed in Algorithm 2 and our adaption of Paterson and Stockmeyer [PS73] to masked polynomials with coefficients in $\mathbb{F}_q$. Table 3 highlights the complexity of evaluating the mappings FM, FU and FV. FU_{1024} and FV_{1024} refer to the mapping FU and FV for the parameter set ML-KEM-1024 where $d_u = 11, \|U\|_\infty \leq 1$ and $d_v = 5, \|V\|_\infty \leq 52$. ML-KEM-512 and ML-KEM-768 share the same values for d_u and d_v but have different *sec* parameters.

Table 3: Complexity of mappings in terms of sensitive multiplications

Mapping F	Complexity of P_F	Total complexity	Occurrences
FM	$58 = 1 + 1 + (2\sqrt{784})$	71	$\times 256$
FU	3	16	$\times 256 \times sec$
FV	$21 = 1 + 1 + (2\sqrt{100} - 1)$	34	$\times 256$
FU_{1024}	2	15	$\times 256 \times 4$
FV_{1024}	$15 = 1 + 1 + (2\sqrt{49} - 1)$	28	$\times 256$

6 Cost-Amortization

Cost-Amortization was first introduced by Wang et al. [WMCS20]. The underlying idea is to mask several secrets within one code word, thus reducing the amount of computations required at the cost of either security or code length. In this section we present a method to swap from a Code-Based Masking using Cost-Amortization to one not using it (and back) when using Code-Based Masking in characteristic 2. We denote an un-amortized Code-Based Masking by $\texttt{Mask}$ and an amortized one by $\texttt{Mask}_{CA}$ for this section.

Remark 9. We assume that $\texttt{Mask}_{CA}$ and $\texttt{Mask}$ both use the same matrices A and $V(\omega)$. They do not share the same G and H matrices in the DSM model however. Thus, when choosing A and V we use $H_{CA}^{\perp}$ to set our desired masking order taking into account Cost-Amortization.

6.1 From Two Code Words to One Amortized

For readability in this section, we omit the $\times A^{-1} \times V(\omega)$ when detailing either $\texttt{Mask}(x)$ or $\texttt{Mask}_{CA}(x, y)$. We have $\texttt{Mask}(x) = (x, \vec{r})$ and $\texttt{Mask}(y) = (y, \vec{r'})$. To

compute $\texttt{Mask}_{CA}(x, y)$ from $\texttt{Mask}(x)$ and $\texttt{Mask}(y)$, we first compute $\texttt{Mask}_{CA}(x, 0)$ and $\texttt{Mask}_{CA}(0, y)$. We then XOR the results.

As we use Reed-Solomon codes, we have the following:

$$P(X) = \texttt{IDFT}(\texttt{Mask}(x)), P(X) = \sum_{i=0}^{k-1} p_i X^i. \tag{30}$$

Evaluating P in u_i allows to recover the i^{th} symbol in $(x, \vec{r})$. We compute the desired mask as follows:

$$\texttt{Mask}(x) \oplus \texttt{Mask}_{CA}(0, P(u_1)) \iff \texttt{Mask}_{CA}(x, r_1) \oplus \texttt{Mask}_{CA}(0, r_1) = \texttt{Mask}_{CA}(x, 0). \tag{31}$$

To compute $\texttt{Mask}_{CA}(0, y)$ from $\texttt{Mask}(y)$, we rely on a similar equation:

$$\text{Given } Q(X) = \texttt{IDFT}(\texttt{Mask}(y)) = \sum_{i=0}^{k-1} q_i X^i, \text{ we set } T(X) = \sum_{i=0}^{k-1} q_i^2 X^i \tag{32}$$

As we are working in characteristic 2, we have the following equation:

$$T(u_1) = T(u_0^2) = \sum_{i=0}^{k-1} q_i^2 (u_0^2)^i = \left(\sum_{i=0}^{k-1} q_i u_0^2 \right)^i = (Q(u_0)^2) = y^2 \tag{33}$$

As we are in a characteristic 2 field, we have $y^2 = y$. Thus, $T(u_1) = y$ and $\texttt{DFT}(T) = \texttt{Mask}_{CA}(T(u_0), y)$. By computing $\texttt{Mask}_{CA}(T(u_0), 0)$, we then have

$$\texttt{Mask}(T(u_0), 0) \oplus \texttt{Mask}(T(u_0), y) = \texttt{Mask}(0, y) \tag{34}$$

6.2 Code-Based Masking De-Amortization

We want to swap from one code word encoding two data to two code words encoding one datum each. For the first datum, this can be trivially performed by applying an un-amortized Refresh:

$$\texttt{Mask}(0) \Leftrightarrow (0, \vec{r}) \times A^{-1} \times V(\omega) \Leftrightarrow \texttt{Mask}_{CA}(0, r_1) \text{ with } \vec{r} \text{ random} \tag{35}$$

Thus, as r_1 is uniformly random,

$$Refresh(\texttt{Mask}_{CA}(x, y)) \Leftrightarrow \texttt{Mask}_{CA}(x \oplus 0, y \oplus r_1) \Leftrightarrow \texttt{Mask}_{CA}(x, r_1') \Leftrightarrow \texttt{Mask}(x) \tag{36}$$

To get the code words for the other data, we can compute $P(X) = \texttt{IDFT}(mask_{CA})$. There exists a polynomial $Q(X)$ such as $Q(X) = P(X^2)$ and thus $Q(U_0) = P(U_0^2) = P(U_1)$. However, the degree of Q is $2k - 2$. To reduce it to $k - 1$ we use similar method to the ones used in the multiplication.

Swapping from $\texttt{Mask}_{CA}$ to $\texttt{Mask}$ and back scales trivially with the amount of secrets encoded within one code word.

6.3 Cost-Amortization in ML-KEM

We investigate about the possible use of Cost-Amortization within ML-KEM. A first application would be the parallel ciphering of several instances of ML-KEM. However, it is also possible to parallelize some parts of ML-KEM. We restricted ourselves to parallelize Code-Based Masking in characteristic 2 and leave the investigation of parallelization of Code-Based Masking in characteristic q as future work.

During the `PKE.Encrypt` procedure, several instances of CBD are computed. We can use Cost-Amortization to parallelize them at an algorithmic level. The seed taken as input of these instances is un-amortized and we use the methodology we just detailed to create amortized masks. All the CBD instances use the same seed, however a nonce is added to the seed as a salt to ensure different results through the `PRF`. Thus, we amortize the seed with itself and the nonce among themselves. This helps in reducing the amount of Keccak instances needed to perform all the CBD instances. The results are de-amortized at the end of the CBD instances, when a conversion to arithmetic is required.

7 Experimental security

7.1 Leakage Assessment

We test the resilience of our design by conducting a statistical leakage detection on physical side-channel measurements. We use a ChipWhisperer Husky to capture power consumption traces. The targeted support is a SAM4S mounted on the CW313 platform, namely the ATSAM4S2A clocking at 7 MHz. As the traces for specific functions can contain a great magnitude of clock cycles, we take only 1 sample per clock cycle.

To detect leakage we rely on the Test Vector Leakage Assessment (TVLA) [GGJR+11]. This testing methodology performs a Welch t-test. We use the fixed vs random approach. The goal is to ensure that the correlation factor between traces captured with a fixed secret and traces captured with a random secret does not exceed a certain threshold. The most commonly used threshold used in the literature is 4.5 and corresponds to an α value of 0.01. However, as our t-tests are performed on a great number of points, we rely on observations made by Ding et al. [DZD+17] to adapt the threshold for each t-test we perform. Otherwise, the probability of observing at least one false positive would be close to 1, thus falsely labelling the target as leaky. We remind the formulae to compute the new α_{TH} and TH threshold values given by Ding et al. [DZD+17]:

$$\alpha_{TH} = 1 - (1-\alpha)^{\frac{1}{N_L}}, \qquad TH = CDF^{-1}_{N(1,0)}(1 - \alpha_{TH}/2). \tag{37}$$

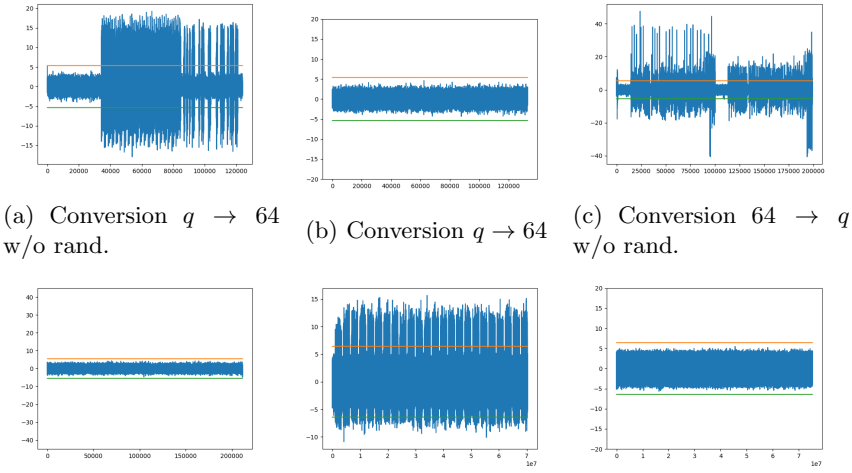
N_L is the number of points in the t-test and $CDF_{N(1,0)}$ the Cumulative Distribution Function of the standard normal distribution. Table 4 uses those formulae to estimate the correct thresholds for each TVLA.

Table 4: Thresholds values for each t-tests and associated Figures

Target	Figure	Number of traces	Number of points	Threshold
Conversion $q \to 64$	3a,3b	500 000	132 766	5.37
Conversion $64 \to q$	3c,3d	500 000	211 622	5.46
Compression	4a,4b	50 000	552 083	5.62
CBD	4c,4d	50 000	652 662	6.05
Keccak	3e, 3f	30 000	75 193 521	6.42

Remark 10. The compression and the CBD in Table 4 are both performed on only one polynomial coefficient.

Our t-test results are summed up in Figures 3 and 4. There are two t-tests per targeted gadgets. One t-test is performed with the randomness turned off, meaning the masks are not randomized. This is denoted by the tag "w/o rand." in our figures' subcaptions. The other t-test is performed with the randomness turned on. This highlights the effectiveness of our design at the first order of masking.



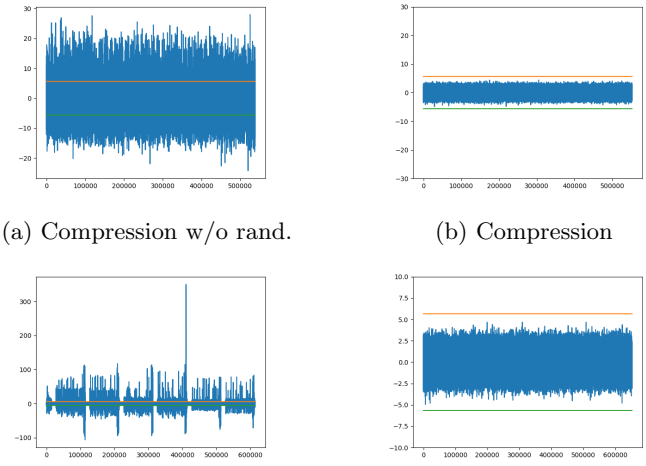
(a) Conversion $q \to 64$ w/o rand. (b) Conversion $q \to 64$ (c) Conversion $64 \to q$ w/o rand.

(d) Conversion $64 \to q$ (e) Keccak w/o rand. (f) Keccak

Fig. 3: TVLA for the conversions and Keccak instances

7.2 Fault Injection Attack Resilience

Similarly to Heinz and Pöppelmann [HP23], we make our FIA resilience claims based on a realistic fault model [KSV13,BBK16]. We consider an attacker capa-



(a) Compression w/o rand. (b) Compression

(c) CBD w/o rand. (d) CBD

Fig. 4: TVLA for the compression and the CBD, both on one polynomial coefficient

ble of performing one precise fault injection out of 3 categories on a coefficient during data processing or storage. The attacker cannot impact the input parameters nor the instruction processing part. We also consider that this attacker cannot perform a fault during the detection/correction process. We leave fault propagation as future work.

The three categories of faults we claim resilience against are:

- *Bit flipping fault*: an attacker can flip or set the values of some bits within a code word element,
- *Random fault*: an attacker can disturb a computation or memory access. As a result the device proceeds a random data. However, we reduce our claim on this fault to the scenarios where the fault occurs within the code word and not on the code word address. We thus consider that the faulted device calls the right code word but some elements of the code word are randomized by this type of faults.
- *Zeroization fault*: an attacker sets a code word to 0.

Our resilience to these faults is a direct consequence of the use of error-correcting codes. Fault detection on a code word $\mathcal{C}$ of $\mathcal{RS}[l, k, l-k+1]$ is performed by computing $c = \texttt{IDFT}(\mathcal{C})$. If the degree of c is not $k-1$, it means some faults have occurred. This detection can be performed with complexity $\mathcal{O}(l \log(l))$ and we can detect as well as correct up to $l-k$ faults.

More details are available in the Carlet et al. [CDGT24] paper we based our masking on.

8 Performances

We test our implementation on a personal laptop and a microcontroller. The laptop is 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 . We have the laptop performances in Table 5, at orders 1 and 3, with Cost-Amortization (CA) or without CA:

Table 5: Performances in milliseconds at different orders on a laptop

Masking order	0	1	1(CA)	3(CA)	3
Key generation	0.03	17.67	20.22	42.32	70.42
Encapsulation	0.03	17.76	24.87	56.83	76.10
Decapsulation	0.03	29.90	31.98	68.47	101.14

Remark 11. Our implementations on laptop and microcontroller are not optimized as they are *PoC* (Proof-of-Concept) implementations in plain C. On the other side, the reference implementation is fully optimized and uses the hard-coded SHA3 in the laptop CPU.

Remark 12. In Table 5, we see that Cost-Amortization gives better performances at order 3 but worse at order 1. We leave a more thorough study on the benefits of using Cost-Amortization in ML-KEM as future work. Our comparison with Bos et al. [BGR+21] on microcontroller will not use Cost-Amortization for this reason.

We also test our design on a microcontroller. We use a ATSAM4S-XPRO board for performances evaluation. Performances are presented in Table 6. For the sake of comparison, we also add in Table 6 the performances of another masked implementation [BGR+21]. However, there are several important points:

- The implementation we compare to is performed on CRYSTALS-Kyber and not ML-KEM. There are thus minor differences.
- Bos et al. [BGR+21] proposed a masked implementation of Kyber768. We evaluate ML-KEM-512. We use a different board, as Bos et al. [BGR+21] perform their evaluation on a STM32F407G. Yet, both boards use the Cortex-M4 technology.
- Our design is not optimized from an implementation point of view. We have yet to use assembly subroutines and other implementation tricks to improve our performances. Our goal in this work is to demonstrate the feasibility and experimental security of our design, while showcasing the challenges we face in using a Code-Based Masking to protect a Post-Quantum KEM.

Remark 13. We also made a slight adjustment compared to the ML-KEM standard [MLK24]. By taking into account some comments made on the FIPS 203

draft, we pre-hash the received ciphertext c before its concatenation with the secret $\vec{z}$ during the KEM Decapsulation. This reduces the size of the input of the sensitive hash function J at the cost of an unmasked call to the hash function H, thus greatly improving performances.

We also add to the comparison in Table 6 the performances of the *pqm4* project as shown in [BGR+21].

8.1 Discussion on Results Compared to the State-of-the-art

The results presented in Table 6 highlight several points:

- Our masking scales better than [BGR+21] with the masking order, thus experimentally verifying its quasilinear nature compared to the quadratic nature of the masking of [BGR+21].
- For the ciphertext comparison, we have a similar order of magnitude in terms of cost to [BGR+21] at the third order of masking. It could be interesting to see how our method will fare when applied to a classical masking as it seems more efficient at higher orders than current state-of-the-art masked ciphertext comparison.
- We perform better in terms of randomness than [BGR+21] at the third order of masking for `PKE.Decrypt`.

Remark 14. Our Code-Based Masking is able of correcting up to j faults at masking order j. A naive way of correcting this much faults with classical masking is to use repetition codes: One can run $2*j+1$ instances of the same classical masking and perform a masked majority vote to correct faults. Comparisons are provided in Figure 5. We can conjecture that asymptotically, our method will perform better at higher orders than the state-of-the-art, both in times and randomness.

8.2 Comparison with Classical Masking

Arithmetic and boolean maskings have limitations at higher orders, due to their quadratic complexity both in time and randomness. Several works have also investigate the security of those masking against different types of attacks at higher order and exposed weaknesses [BCPZ16,BS21]. Code-Based Masking seems to suffer less from those issues as its complexity in both time and randomness is quasilinear. However, its better resilience at higher orders compared to other masking is still only a conjecture as of writing this work.

9 Conclusion

In this paper we propose a masked implementation of ML-KEM using Code-Based Masking. Using the work from Carlet et al. [CDGT24], we propose specific

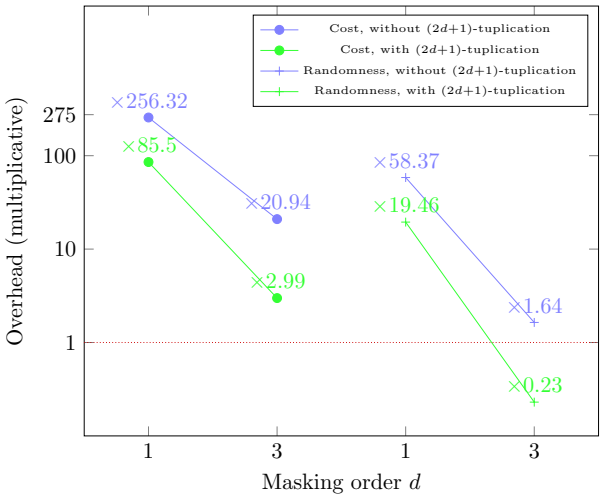


Fig. 5: Multiplicative overheads compared to [BGR+21]

Table 6: Performances in 10^3 cycles at the first and third orders of masking on a microcontroller

Operation	*pqm4*	[BGR+21]			[**This work**]		
Masking Order	0	1	3 (vs 1st order)		1	3 (vs 1st order)	
`crypto_kem_dec`	882	3116	115 481	(×37)	798 715	2 418 901	(×3)
`indcpa_dec`	–	174	9288	(×53.4)	240 827	346 014	(×1.4)
`Conversion`	–	–	–	–	6503	52 176	(×8)
`hashg`	–	118	2659	(×22.5)	34 708	89 628	(×2.6)
`indcpa_enc`	–	2196	30 938	(×14.1)	450 185	1 759 110	(×3.9)
`comparison`	–	462	72 568	(×157.1)	31 926	83 791	(×2.6)
`kdf`	–	14	14	(×1)	328	992	(×3)
#randombytes	–	12 072	2 434 170	(×201.6)	704 631	3 998 745	(×5.7)
`indcpa_enc`	676	2196	30 838	(×14.1)	450 185	1 759 110	(×3.9)
`poly_arithm`	–	301	653	(×2.2)	6873	7852	(×1.1)
`poly_getnoise`	–	1384	29 347	(×21.2)	443 313	1 751 258	(×4)
#randombytes	–	7030	562 974	(×80.1)	527 452	3 448 926	(×6.5)
`indcpa_dec`	64	174	9288	(×53.4)	240 827	346 014	(×1.4)
`unpack`	–	23	36	(×1.6)	150	413	(×2.8)
`poly_arithm`	–	89	149	(×1.7)	3778	4920	(×1.3)
`compress`	–	61	9102	(×149.2)	236 902	340 641	(×1.4)
#randombytes	–	640	201 984	(×315.6)	48 256	117 504	(×2.4)

Remark 15. Due to the intellectual property regulation in place in our working environment, we are not able to share the source code of our *PoC* implementations at the moment.

codes to mask ML-KEM in Section 3. We provide improved security against Fault Injection Attacks (FIA) on ML-KEM compared to the current state-of-the-art [HP23] by being able to correct faults. In Section 4 we introduced a methodology to perform conversions between two different Code-Based Masking. To address the non-linear functions of ML-KEM, we propose new generic solutions. We adapt an evaluation method from Paterson and Stockmeyer [PS73] to securely evaluate a polynomial in characteristic other than 2 in Section 5. We use this evaluation method alongside Fermat's little theorem to secure the message compression as well as the ciphertexts comparison. The security of our design is experimentally verified using a Welsh's t-test and results are presented in Section 7. Our performances' comparison in Section 8 shows that, while we have higher costs in time and randomness at the first masking order compared to the state-of-the-art [BGR+21], at higher orders our method will be asymptotically more efficient with the added benefit of correcting/detecting more faults. We best the septuplication[6] of the third masking order of Bos et al. [BGR+21] in terms of randomness needs.

Our method for the masked ciphertext comparison seems to fare better with masking order scaling than the state-of-the-art [BGR+21] as we are less reliant on conversions. However this hypothesis has to be validate by using the same masking for comparison. Finally, with Deep Learning Side Channel Analysis [DNGW23,WBD24] able of breaching state-of-the-art masked ML-KEM at low orders, it would be interesting to study the impact of the structure of Code-Based Masking on such attacks.

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 France SAS 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) which take part in the France 2030 vision. In this, Hensoldt France SAS is a part of the X7-PQC project in partnership with Secure-IC, Télécom Paris and Xlim.

[6] Necessary to correct 3 faults

References

AAC+22. Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the third round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2022.

BBD+16. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.

BBK16. Nina Bindel, Johannes Buchmann, and Juliane Krämer. Lattice-based signature schemes and their sensitivity to fault attacks. In *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*, pages 63–77. IEEE Computer Society, 2016.

BC22. Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. *IACR TCHES*, 2022(4):553–588, 2022.

BCC+14. Julien Bringer, Claude Carlet, Hervé Chabanne, Sylvain Guilley, and Houssem Maghrebi. Orthogonal direct sum masking: A smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks. Cryptology ePrint Archive, Report 2014/665, 2014. `https://eprint.iacr.org/2014/665`.

BCPZ16. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer, Heidelberg, August 2016.

BEF+23. Sebastian Berndt, Thomas Eisenbarth, Sebastian Faust, Marc Gourjon, Maximilian Orlt, and Okan Seker. Combined fault and leakage resilience: Composability, constructions and compiler. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 377–409. Springer, Heidelberg, August 2023.

BGR+21. Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and higher-order implementations. *IACR TCHES*, 2021(4):173–214, 2021. `https://tches.iacr.org/index.php/TCHES/article/view/9064`.

BS21. Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms. *IACR TCHES*, 2021(3):202–234, 2021. `https://tches.iacr.org/index.php/TCHES/article/view/8973`.

CCJ+16. Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray A Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology . . . , 2016.

CD23. Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 423–447. Springer, Heidelberg, April 2023.

CDGT24. Claude Carlet, Abderrahman Daif, Sylvain Guilley, and Cédric Tavernier. Quasi-linear masking against SCA and FIA, with cost amortization. *IACR TCHES*, 2024(1):398–432, 2024.

CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Heidelberg, August 1999.

CPRR14. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 410–424. Springer, Heidelberg, March 2014.

DM21. Jeroen Delvaux and Santos Merino Del Pozo. Roulette: Breaking Kyber with diverse fault injection setups. Cryptology ePrint Archive, Report 2021/1622, 2021. `https://eprint.iacr.org/2021/1622`.

DNGW23. Elena Dubrova, Kalle Ngo, Joel Gärtner, and Ruize Wang. Breaking a fifth-order masked implementation of crystals-kyber by copy-paste. In Masayuki Fukumitsu and Shingo Hasegawa, editors, *Proceedings of the 10th ACM Asia Public-Key Cryptography Workshop, APKC 2023, Melbourne, VIC, Australia, July 10-14, 2023*, pages 10–20. ACM, 2023.

Dwo15. Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. *NIST FIPS*, 2015.

DZD+17. A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. 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 105–122. Springer, 2017.

FO99. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 537–554. Springer, Heidelberg, August 1999.

FP22. Tako Boris Fouotsa and Christophe Petit. A new adaptive attack on SIDH. In Steven D. Galbraith, editor, *CT-RSA 2022*, volume 13161 of *LNCS*, pages 322–344. Springer, Heidelberg, March 2022.

GGJR+11. Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

HKL+22. Daniel Heinz, Matthias J. Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Amber Sprenkels. First-order masked Kyber on ARM Cortex-M4. Cryptology ePrint Archive, Report 2022/058, 2022. `https://eprint.iacr.org/2022/058`.

HP23. Daniel Heinz and Thomas Pöppelmann. Combined fault and DPA protection for lattice-based cryptography. *IEEE Trans. Computers*, 72(4):1055–1066, 2023.

ISW03. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

JAC+22. David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev,

David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. SIKE. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions`.

KCS+24. Suparna Kundu, Siddhartha Chowdhury, Sayandeep Saha, Angshuman Karmakar, Debdeep Mukhopadhyay, and Ingrid Verbauwhede. Carry your fault: A fault propagation attack on side-channel protected LWE-based KEM. *IACR TCHES*, 2024(2):844–869, 2024.

Koc96. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996.

KSV13. Dusko Karaklajic, Jörn-Marc Schmidt, and Ingrid Verbauwhede. Hardware designer's guide to fault attacks. *IEEE Trans. Very Large Scale Integr. Syst.*, 21(12):2295–2306, 2013.

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010.

MLD24. Module-lattice-based digital signature. National Institute of Standards and Technology, NIST FIPS PUB 204, U.S. Department of Commerce, August 2024.

MLK24. Module-lattice-based key-encapsulation mechanism. National Institute of Standards and Technology, U.S. Department of Commerce, August 2024.

MMP+23. Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. A direct key recovery attack on SIDH. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 448–471. Springer, Heidelberg, April 2023.

MP22. M Mosca and M Piani. 2021 quantum threat timeline report global risk institute. *Global Risk Institute*, 2022.

OSPG18. Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure masked Ring-LWE implementations. *IACR TCHES*, 2018(1):142–174, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/836`.

PS73. Mike Paterson and Larry J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.

Rob23. Damien Robert. Breaking SIDH in polynomial time. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 472–503. Springer, Heidelberg, April 2023.

RS02. Eric M Rains and Neil JA Sloane. Self-dual codes. *arXiv preprint math/0208001*, 2002.

RV13. Arnab Roy and Srinivas Vivek. Analysis and improvement of the generic higher-order masking scheme of FSE 2012. Cryptology ePrint Archive, Report 2013/345, 2013. `https://eprint.iacr.org/2013/345`.

SAB+22. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`.

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.

SLH24. Stateless-hash-based digital signature algorithm. National Institute of Standards and Technology, NIST FIPS PUB 203, U.S. Department of Commerce, August 2024.

WBD24. Ruize Wang, Martin Brisfors, and Elena Dubrova. A side-channel attack on a higher-order masked CRYSTALS-kyber implementation. In Christina Pöpper and Lejla Batina, editors, *ACNS 24, Part III*, volume 14585 of *LNCS*, pages 301–324. Springer, Heidelberg, March 2024.

WMCS20. Weijia Wang, Pierrick Méaux, Gaëtan Cassiers, and François-Xavier Standaert. Efficient and private computations with code-based masking. *IACR TCHES*, 2020(2):128–171, 2020. `https://tches.iacr.org/index.php/TCHES/article/view/8547`.

WZ88. Yao Wang and Xuelong Zhu. 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.