Advances in Reed-Solomon Code-Based Masking and Application to ML-KEM

Pierre-Augustin Berthet[1,2], Yoan Rougeolle[1], Cédric Tavernier[1] and Laurent Sauvage[2]

[1] Hensoldt France SAS, Plaisir, France
[2] Institut Polytechnique de Paris, Télécom Paris, LTCI, Palaiseau, France

Abstract. Physical attacks such as Side-Channel Analysis (SCA) or Fault Injection Attacks (FIA) can recover sensitive data from cryptographic primitives otherwise thought theoretically secure. To counter such threats, generic countermeasures such as masking are studied. In this work, we provide some advances on one particular form of masking, Reed-Solomon Code-Based Masking (RS-CBM). Although its application to the AES primitive with Boolean logic has been investigated, we propose arithmetic gadgets and constrained conversions from arithmetic to boolean logic and back. We also investigate Cost-Amortisation (CA), a method to encode several sensitive data into one masked code word, and propose techniques to swap between an un-amortised masking and an amortised one. 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. Finally, we apply our gadgets to a post-quantum Key Encapsulation Mechanism (KEM), ML-KEM. Notably, we propose a full arithmetisation of the masked calculations of the message compression and of the ciphertext comparison of ML-KEM.

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

1 Introduction

Since the late 1990s and the publication of Kocher on Side-Channel Analysis (SCA)[Koc96], physical attacks have attempted to take advantage of physical leakages or faults injected within the implementation to recover sensitive data. One of the most efficient and proven countermeasures against SCA is masking[CJRR99]. The core idea is to avoid manipulating sensitive data but instead random *shares* of it. The shares are reassembled once the computations have been performed. The shares are a combination of sensitive data and a number of random variables called *masks*. We denote by *sharing* the set of shares that mask a sensitive value. The masking order is determined by the minimum number of shares that an attacker needs to recover the secret data. High-order masking means better security against differential attacks, but it generally comes at the cost of worse performance 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 Bringer et al.[BCC+14] Direct Sum Masking (DSM), namely a Code-Based Masking adapted from Carlet et al.[CDGT24].

E-mail: berthet@telecom-paris.fr (Pierre-Augustin Berthet), yoan.rougeolle@hensoldt.net (Yoan Rougeolle), cedric.tavernier@hensoldt.net (Cédric Tavernier), laurent.sauvage@telecom-paris.fr (Laurent Sauvage)

The idea is to add structure and redundancy to the shares. The use of a Reed-Solomon error-correcting code allows for fast detection and correction of faults.

As an application example of our advances on Code-Based Masking, we focus on Post-Quantum Cryptography. 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 such as signatures and Public Key Encryption (PKE) by taking advantage of the Shor quantum algorithm[Sho94]. The National Institute for Standards and Technology (NIST) launched a Post-Quantum Cryptography (PQC) standardisation process in 2016[CCJ+16]. This international competition aims to standardise digital signature and Key Encapsulation Mechanism (KEM) protocols secured against quantum and classical computers. Three signatures and one KEM were selected[AAC+22], while another KEM was selected in a fourth round as an alternative[ABC+25]. At the time of writing this work, three out of five 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].

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 example, Heinz et al. [HKL+22] proposed the first open source implementation of a masked Kyber on the microprocessor while relying on the work of Oder et al. [OSPG18] on previous lattice-based primitives. Bos 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 combined fault and DPA protection for lattice-based cryptography. However, they only secured the linear parts of the algorithm and were unable to correct the faults. Fault attacks against masked implementations of ML-KEM are a real concern, with work by Delvaux [DM21] and Kundu et al. [KCS+24] successfully using FIA to break masked implementations of ML-KEM on microcontrollers. Thus, there is a clear need for solutions with better resilience to combined attacks using SCA and FIA. Our work is one possible solution with wider possibilities for error detection and an error-correcting capability that does not yet exist to this day in the state-of-the-art literature regarding masked ML-KEM to our knowledge. The correction capacity can also be useful for deployment in hostile or radiation-heavy environments, e.g. satellite applications where cosmic radiation can affect the computations.

1.1 Our Contributions

Carlet et al. [CDGT24] applied Reed-Solomon Code-Based Masking (RS-CBM) to the AES and proved the t-probing security of their gadgets. This symmetric cryptography primitive uses Boolean logic. In this paper, we propose several extensions of their work and study its application to a post-quantum KEM.

We present the addition, scaling, and multiplication gadgets for RS-CBM. We provide stronger security proofs, both formal and experimental. We use the t-Strong Non-Interference (SNI) criteria of Barthe et al. [BBD+16] for the formal part. For experimentation, a Test Vector Leakage Assessment (TVLA) [GGJR+11] is performed on isolated gadgets using a Chipwhisperer Husky and a SAM4S microcontroller on the CW-313 target board. This TVLA is performed on several orders of masking.

Among the novelties of our work is also the study of a constrained conversion between two types of RS-CBM (arithmetic and boolean). The constraint is that our method is only

applicable if the masked value we intend to convert is a mask of 1 or 0.

Another novelty involves Cost-Amortisation from Wang et al.[WMCS20], a method to mask several sensitive data into one sharing. In this work, we propose a method to swap between several "un-amortised" sharings and a fully "amortised" one, and back. However, this only applies when the sharings use RS-CBM defined over finite fields of characteristic 2.

Finally, we apply our advances to ML-KEM[MLK24], a post-quantum KEM, as an example of application. We adapt the calculations of the non-linear functions of ML-KEM to the lack of unconstrained conversions in our RS-CBM. As a result, another contribution of this work is the proposal of fully arithmetised computations for message compression and ciphertext comparison in ML-KEM.

The paper is structured as follows. In Section 2, we introduce the notation, formal and experimental security notions for masking, and the ML-KEM algorithm. New advances on Reed-Solomon Code-Based Masking (RS-CBM) are presented in Section 3. In particular, we present constrained conversions in Section 3.5 and advances on Cost-Amortisation in Section 3.6. We use ML-KEM as an application example of our advances in Section 4. Section 5 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 ($\texttt{DFT}$) and its inverse ($\texttt{IDFT}$ 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

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

We perform $\texttt{DFT}^{-1}$ by composing the $\texttt{DFT}$ 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 $\mathcal{RS}$ of length l and dimension k that:

- If $\mathcal{RS}$ is MDS, then $\mathcal{RS}^{\perp}$ is MDS where $\mathcal{RS}^{\perp}$ is the code defined by kernel(R);
- If $\mathcal{RS}$ 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 Formal Security Proofs

2.2.1 $t-$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 physical accesses to the device under attack using measurement of t probes and are 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.

2.2.2 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 randomised function $\mathtt{S}$: $\mathbb{F}_q^{|\chi'|} \to \mathbb{F}_q^{|\mathcal{P}|}$. A distinguisher is a randomised function $\mathtt{D}$: $(\mathbb{F}_q^{|\mathcal{P}|}, \mathbb{F}_q^{|\chi|}) \to \{0, 1\}$. 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 for securely composing 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 to be $t-$Strong Non-Interference secure ($t-$SNI) if and only if any set of at most t intermediate variables whose t_1 probes on the internal variables (*id est* intermediate variables except the output's ones) and t_2 probes on output variables can be perfectly simulated from at most t_1 shares of each input.

A consequence, we have the following implications:

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

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.

2.3 Test Vector Leakage Assessment

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 widely used threshold used in the literature is 4.5 and corresponds to an α value of 0.01. However, since the t-tests in this paper are performed on a large number of points, we rely on observations made by Ding et al. [DZD+17] to adapt the threshold for each t-test. Otherwise, the probability of observing at least one false positive would be close to 1, thus falsely labelling the target as leaky. We recall the formulae for computing the new threshold values α_{TH} and TH given by Ding et al. [DZD+17]:

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

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.

We test the resilience of our design by performing a statistical leak detection on physical side-channel measurements. A ChipWhisperer Husky is used to capture power consumption traces. The targeted support is an SAM4S mounted on the CW313 platform, namely the ATSAM4S2A clocking at 7 MHz. The sampling rate is 8 samples per clock cycle. For each gadget, 100 000 traces are taken. Table 1 uses these formulae to estimate the correct thresholds for each TVLA we perform in this paper.

Table 1: Thresholds values for each t-tests and associated Figures

Target	Figure	Number of points	Threshold
XOR order 1	1b	1304	4.47
XOR order 3	1c	1776	4.53
ADD order 1	2b	1752	4.53
ADD order 3	2c	2328	4.60
AND order 1	5b	3272	4.66
AND order 3	5c	12 786	4.93
MUL order 1	6b	13 224	4.94
MUL order 3	6c	20 304	5.02
Conversion $\mathbb{F}_{16} \to \mathbb{F}_q$	7a	284 672	5.51
Conversion $\mathbb{F}_q \to \mathbb{F}_{16}$	7b	272 536	5.50
Split	8a	13 616	4.93
Merge	8b	12 768	4.90

The source code of our gadgets is available in the following Github repository:

https://github.com/carot250704/reedsolomonmasking

2.4 ML-KEM

ML-KEM or FIPS 203 [MLK24] is the first post-quantum KEM standard by 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 the 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, that is, $q = 3329$ and the length of the message $n = 256$. 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.

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

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

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

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

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

Remark 2. The compression functions are lossy:

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

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 used on public data in addition to its usage on the message documented in Remark 1. It does not require to be masked.

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

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

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

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

`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 Encryption, Decryption, and Decapsulation [MLK24]

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

Sub-function `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 $m = Compress_q(v - \vec{s^t} \cdot \vec{u}, 1)$

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.

Remark 4. PKE.Encrypt always results 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 leads to a completely different result out of the re-encapsulation, hence the CCA-security.

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

3 Reed-Solomon Code-Based Masking

Arithmetic or boolean masking can be seen through the scope of code theory as punctured parity codes. Thus, they do not have any detection or error correction capabilities. On the other hand, Direct-Sum Masking (DSM) is more generic and allows the use of redundancy

structures to provide these capabilities. In this paper, we use a specific type of DSM introduced by Carlet et al. [CDGT24]: Reed-Solomon Code-Based Masking.

In this section, we present the different gadgets in a similar manner: first, we explain the rational behind the gadget, then the gadget is detailed. A study of the security of the gadget is provided from a formal and an experimental point of view.

3.1 Masking gadget

To construct a d order masking gadget, the DSM method uses the following function to mask a vector $\vec{x} \in \mathbb{F}_q^\tau$:

$$\forall \vec{x} \in \mathbb{F}_q^\tau, \vec{r} \text{ random} \in \mathbb{F}_q^{d-\tau}, \texttt{Mask}_{DSM}(\vec{x}, \vec{r}) = \vec{x} \cdot G + \vec{r} \cdot H \tag{5}$$

The parameter τ refers to the quantity of information to be masked. Most of the existing methods are built to mask a single symbol whereas masking several symbols is an added value of our method and is described as *cost amortization* in section 3.6. In Equation 5, 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 DSM framework with Reed-Solomon codes. This allows the use of the Discrete Fourier Transform (DFT) in many operations within this Code-Based Masking. Fault detection and masked multiplications are performed in quasilinear time.

In this paper, we consider the same masking technique that is adapted to fields of odd characteristics. These techniques are also inspired by the results of Goudarzi, Joux and Rivain [GJR18] and Goudarzi et al. [GPRV21].

This method works as follows: two disjoint sets $\{\alpha, \dots, \alpha^d\} \in \mathbb{F}_q^d$ and $\{1, \omega, \dots, \omega^{l-1}\} \in \mathbb{F}_q^l$ with $d = \lfloor l/2 \rfloor$ are considered, then a vector $\vec{r} \in \mathbb{F}_q^{d-\tau}$ is picked randomly and the values $x_0, \dots, x_{\tau-1}, r_0, \dots, r_{d-\tau-1}$ are interpolated over $\{\alpha, \dots, \alpha^d\}$ providing a degree $d-1$ polynomial P. The mask of x is obtained by evaluating P over $\{1, \omega, \dots, \omega^{l-1}\}$. A linear algebra interpretation of the `Mask` is done by the following:

$$\texttt{Mask}(x) = (x, \vec{r}) \cdot A^{-1} \cdot V(\omega) \tag{6}$$

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

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 5. The matrix $V(\omega)$ is a Vandermonde matrix. 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, setting $(u_j)^i = A_{i,j}$, we see that P satisfies $P(u_i) = c_i$.

3.1.1 Formal security of the masking gadget

We first recall the notion of a 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:*

$$\begin{aligned} t &= \max\{i \in \{0,\dots,l\} : \forall w \in \{\mathcal{C}_{H^\perp}\}_i, Gw^T = 0^T\}, \\ &= d - 1 + \max\{i \in \{0,\dots,l-d+1\} : \forall w \in \{\mathcal{C}_{H^\perp}\}_{d-1+i}, Gw^T = 0^T\}. \end{aligned}$$

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. □

3.1.2 Experimental security

Throughout this section, experimental verification of the hardware security of the gadgets is provided through TVLAs. We use the following fields and codes to test each gadget:

- For "arithmetic" gadgets, we use $\mathbb{F}_{3329}$. In the first order of masking, the code is a $[4, 2, 3]$ Reed-Solomon code. In the third order of masking, the code is a $[8, 4, 5]$ Reed-Solomon code.
- For "Boolean" gadgets, the code is a $[3, 2, 2]$ Reed-Solomon code defined over $\mathbb{F}_{16}$ for the first order of masking while for the third order, we use a $[7, 4, 4]$ Reed-Solomon code defined over $\mathbb{F}_{64}$.

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:

$$\texttt{AddMask}(x, x') = \texttt{Mask}(x + x') = \vec{z} + \vec{z'} \tag{7}$$

$$\texttt{ScalMask}(\lambda, x) = \texttt{Mask}(\lambda \cdot x) = \lambda \cdot \vec{z} \tag{8}$$

Remark 6. The XOR gadget is similar to AddMask. In terms of complexity, the XOR gadget costs l XOR, AddMask costs l additions and ScalMask costs l scalar multiplications.

3.2.1 Formal security of the addition and scaling

We use Theorem 3 from [WMCS20], which states 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 *(Equation 7) with inputs $\vec{z} = \texttt{Mask}(x)$, $\vec{z'} = \texttt{Mask}(y)$ and output $\vec{w}$ ensures that $\texttt{Unmask}(\vec{w}) = x + y$, and* AddMask *is $t-$NI for any positive integer t such that $t \leq (d-\tau)$.*

Proof. ([WMCS20]). By construction, we have

$$\texttt{Unmask}(\vec{w}) = \texttt{Unmask}(\vec{z} + \vec{z'}) = \texttt{Unmask}(\texttt{Mask}(x) + \texttt{Mask}(y)) = x + y$$

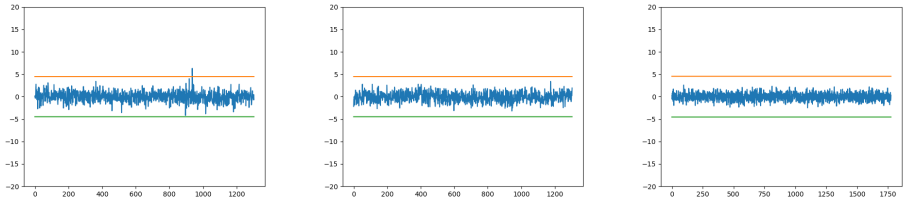
and we get the correctness. Since the operations are performed independently on each share, any probe (internal and output) in `AddMask` has one of the following three forms: $\vec{z}[i], \vec{z'}[i]$ or $\vec{z}[i] + \vec{z'}[i]$ where $i \in [1..l]$. Let I be the indexes appearing in the t probes. By construction $|I| \leq t$, thus $\vec{z}[I]$ and $\vec{z'}[I]$ 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: $Pr[D(C_{\mathcal{P}}(\vec{z}, \vec{z'}), \vec{z}, \vec{z'}) = 1] = Pr[D(S(\vec{z}[I], \vec{z'}[I]), \vec{z}, \vec{z'}) = 1]$,for any distinguisher D. □

Theorem 2. *The gadget* `ScalMask` *(Equation 8) 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. □

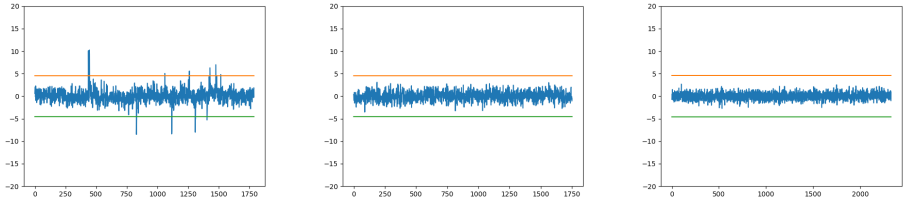
3.2.2 TVLA for the XOR and the Addition

The security of the `AddMask` gadget is also experimentally verified by a TVLA. We also verify the experimental security of the XOR gadget from [CDGT24]. In Figures 1 and 2, several Welsh t-tests are performed in different settings: a first order t-test on the gadget with randomness turned off to highlight potential leakage points, a first order t-test on the first order gadget, and finally a third order t-test on the third order gadget.



(a) XOR w/o rand. (b) XOR order 1 (c) XOR order 3

Figure 1: TVLA of the XOR gadget in three different settings



(a) ADD w/o rand. (b) ADD order 1 (c) ADD order 3

Figure 2: TVLA of the Addition gadget in three different settings

3.3 Refresh gadget

The `Refresh` gadget, inspired by Wang et al. [WMCS20], is performed by adding to the masked value an encoding of 0:

$$\texttt{Refresh}(\texttt{Mask}(x)) = \texttt{Mask}(x) + \texttt{Mask}(0) \tag{9}$$

Theorem 3. *The refresh gadget* `Refresh` *(Equation 9) is $(d - \tau)-$SNI secure.*

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

3.4 Multiplication

We first recall the multiplication gadget from [CDGT24] and adapt it to a case with characteristic $\neq 2$. Then we propose a slightly adjusted construction to guarantee better formal security properties.

If $\vec{z} = \texttt{Mask}(x)$ and $\vec{z}\,' = \texttt{Mask}(y)$, then

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

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 degree $2k-2$ polynomial, which satisfies $C(u_i) = P_x(u_i) \times P_y(u_i) = x_i y_i$ for i in $\{0, \ldots, \tau-1\}$. Now we build a degree $k-1$ polynomial $D(X)$ that satisfies $D(u_i) = C(u_i)$ for all $0 \leq i \leq \tau - 1$. For this polynomial reduction, we introduce the polynomials

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

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

$$\begin{aligned} D(X) = c_0 + c_1 X + \cdots + c_{k-1} X^{k-1} \quad &+ \quad (c_k X + \cdots + c_{2k-\tau-1} X^{k-\tau}) \sum_{j=1}^{\tau} U_j(X) \\ &+ \quad \sum_{i=1}^{\tau-1} c_{2k-\tau-1+i} \sum_{j=1}^{\tau} U_j(X) u_j^{k-\tau+i}, \end{aligned}$$

which satisfies $D(u_i) = C(u_i) = x_i y_i$ of $i \in \{0, \ldots, \tau-1\}$. Hence, we get:

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

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

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

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

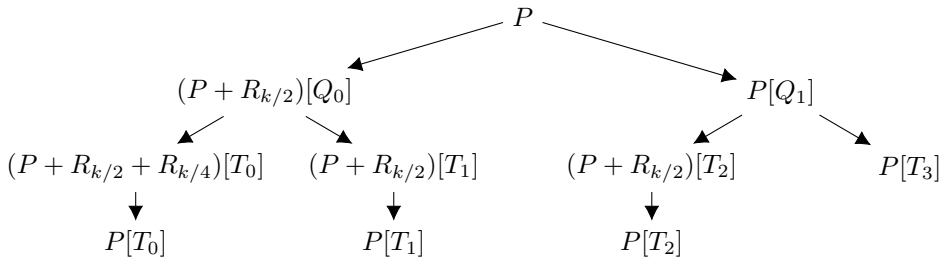


Figure 3: Tree for the DFT

3.4.1 Formal security and fault resilience of the multiplication

The formal security of the Boolean multiplication gadget in Carlet et al. [CDGT24] Code-Based Masking is laid out by Goudarzi et al. [GPRV21]. In their work, they make the assumption of the $t-$probing security of the DFT and prove that their multiplication gadget is $t/2-$probing secure (see [GPRV21], Lemma 2) due to the necessary polynomial reduction. In this section, we provide a $(d-\tau)-$SNI secure DFT as well as a new construction which does not require the polynomial reduction we described in the previous section.

First, we detail the tree decomposition for the DFT in Figure 3. We adopt this peculiar structure to ensure Theorem 4 on the $(d-\tau)-$SNI security of this DFT. In Figure 3, we use the following notations: for readability, we omit (X) in $P(X)$, and also 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 according to the construction of Wang and Zhu [WZ88]. Polynomials R_i are randomised polynomials of degree i. The three instances of R_i within the tree are randomised and different.

Theorem 4. *The* DFT *described in Figure 3 is* $(d-\tau)-$*SNI secure.*

Proof. The tree in Figure 3 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 following morphism is an isomorphism:

$$\begin{aligned} \varphi : \ \frac{\mathbb{Z}[X]}{Q_0 Q_1} &\mapsto \frac{\mathbb{Z}[X]}{Q_0} \times \frac{\mathbb{Z}[X]}{Q_1} \\ P &\to (P[Q_0], P[Q_1]). \end{aligned}$$

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 to retain the error-correcting capability of the masking throughout the multiplication computation, we take inspiration from the work of Berndt et al. [BEF+23] and make a new construction for better security. The multiplication is described in Figure 4. 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

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

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

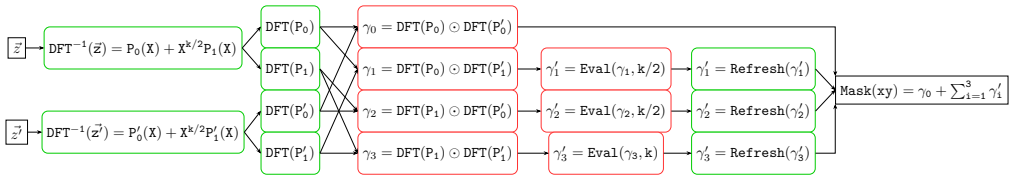


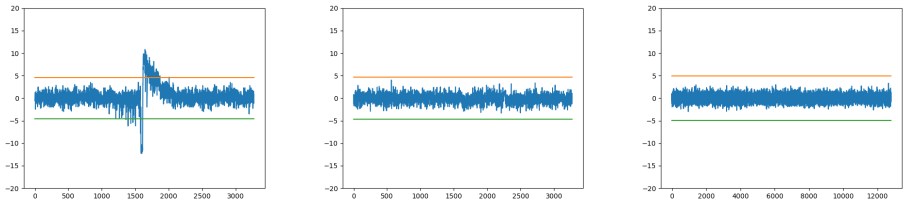
Figure 4: Multiplication gadget with a NI and SNI color code

Theorem 5. *The multiplication gadget described in Figure 4 is $(d-\tau)-$SNI secure.*

Proof. In this proof, we use the composition lemma 1. We justify the colour coding for each block, with green blocks or functions being $(d-\tau)-$SNI and red ones being $t-$NI.
Due to the $(d-\tau)-$SNI security of the DFT proved in Theorem 4, The inverse DFT is $(d-\tau)-$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 to γ_0. 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. The wire tied to γ_0 starts with a $(d-\tau)-$SNI secure gadget. 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. □

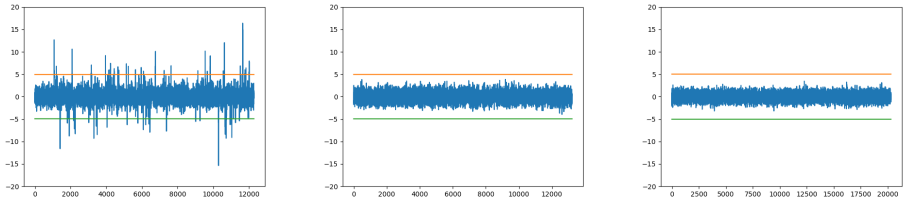
3.4.2 TVLA for the AND and the Multiplication

Similarly to the experimental security check performed on the XOR and AddMask gadgets, several TVLA are performed on the multiplication gadget in Figure 6 and on the AND gadget from [CDGT24] in Figure 5. We use the same approach, with a Welsh t-test with randomness turned off, one at the first order and the last one at the third.



(a) AND w/o rand. (b) AND order 1 (c) AND order 3

Figure 5: TVLA of the AND gadget in three different settings



(a) MUL w/o rand. (b) MUL order 1 (c) MUL order 3

Figure 6: TVLA of the Multiplication gadget in three different settings

3.5 Constrained Conversions

Some primitives alternate between arithmetic calculations and the use of boolean logic. To secure the computation of non-linear functions, Boolean formula can also be used instead of arithmetic. However, while conversions between arithmetic and boolean masking and back are well documented in the literature, there is no prior work on the matter of masked conversions in the case of Code-Based Masking up to our knowledge. In this subsection, we detail how to perform such conversions in the specific case of conversions between masks of 0 or 1.

3.5.1 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,

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

We also have an inverse relation:

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

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

Remark 7. If the sensitive datum is not 0 or 1, it is necessary to obtain its binary decomposition before performing the conversion. This can be performed securely with Lagrange's interpolations. However, this is costly, with an estimated complexity of $\lceil log(q) \rceil \times (2\lfloor \sqrt{q} \rfloor - 1)$ masked multiplications in the worst case[1]. As a result, $\lceil q \rceil$ conversions must be performed.

The overall strategy consists of interpolating the masking operations from one field with the operations of the other field. This allows us 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 $\mathbb{F}_{2^b}$.

3.5.2 From arithmetic to boolean logic

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

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

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

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

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

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

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 perform:

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

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 $\texttt{Unmask}_q(S) = \texttt{Mask}_2(x)$.

3.5.3 From boolean logic to arithmetic

The strategy is similar to the previous conversion except for 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:

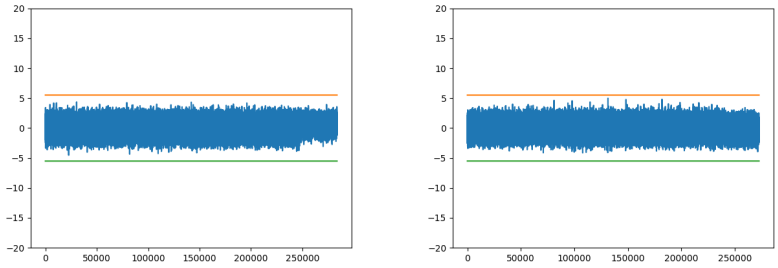
$$S = \begin{pmatrix} \texttt{Mask}_2(g_0^{(0)}) & \ldots & \texttt{Mask}_2(g_{l_q-1}^{(0)}) \\ \ldots & \ldots & \ldots \\ \texttt{Mask}_2(g_{l_q-1}^{(11)}) & \ldots & \texttt{Mask}_2(g_{l_q-1}^{(11)}) \end{pmatrix} +_2 \begin{pmatrix} h_0^{(0)} \otimes \vec{m} & \ldots & h_{l_q-1}^{(0)} \otimes \vec{m}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

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

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

3.5.4 TVLA for the Constrained Conversions

The security of the constrained conversions is experimentally validated with a TVLA. In Figure 7, a Welsh t-test is performed for each constrained conversion at the first order of masking.



(a) Conversion $\mathbb{F}_{16} \to \mathbb{F}_q$ (b) Conversion $\mathbb{F}_q \to \mathbb{F}_{16}$

Figure 7: TVLA of the Constrained Conversions

3.6 Cost-Amortisation

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

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. Thus, when choosing A and V we use $H^{\perp}_{CA}$ to set our desired masking order taking into account Cost-Amortisation.

For readability in this section, we omit the $\times A^{-1} \times V(\omega)$ when detailing $\texttt{Mask}(x)$ or $\texttt{Mask}_{CA}(x, y)$.

3.6.1 From Two Code Words to One Amortised: Merge

This gadget is called the Merge gadget. We have $\texttt{Mask}(x) = (x, \vec{r})$ and $\texttt{Mask}(y) = (y, \vec{r'})$. To compute $\texttt{Mask}_{CA}(x, y) = Merge(\texttt{Mask}(x), \texttt{Mask}(y))$ 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:

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

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

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

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

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

As we are working in characteristic 2, and by construction $u_1 = u_0^2$, we have the following equation:

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

As we mask in Kyber only the elements 0 or 1 at this level of conversion, then 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

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

3.6.2 From One Amortised Code Words to Two Un-amortised: Split

We want to swap from one code word that encodes two data to two code words encoding one datum each. This is the Split gadget such as $Split(\texttt{Mask}_{CA}(x, y)) = \{\texttt{Mask}(x), \texttt{Mask}(y)\}$. For the first datum, this can be trivially performed by applying an un-amortised Refresh:

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

Thus, as r_1 is uniformly random,

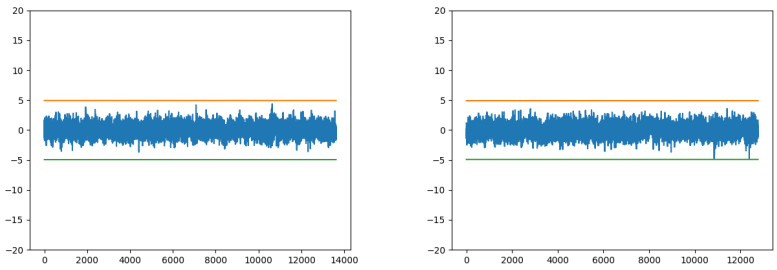
$$Refresh(\texttt{Mask}_{CA}(x, y)) = \texttt{Mask}_{CA}(x \oplus 0, y \oplus r_1) = \texttt{Mask}_{CA}(x, r_1') = \texttt{Mask}(x)$$

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 a method similar to the one used in multiplication.

The transformation of $\texttt{Mask}_{CA}$ to $\texttt{Mask}$ and back scales trivially with the amount of secrets encoded within a code word.

3.6.3 TVLA for the Split and Merge Gadgets

The security of the split and merge gadgets is experimentally verified through a TVLA. A Welsh t-test is performed for each gadget in Figure 8.



(a) Split (b) Merge

Figure 8: TVLA for the Split and Merge gadgets

3.7 Fault Injection Attack Resilience

In this subsection, we centralised the analysis of the resilience of our gadgets to Fault Injection Attacks. Similarly to Heinz and Pöppelmann [HP23] for their FIA-resilient design for ML-KEM, we make our FIA-resilience claims based on a realistic fault model

[KSV13, BBK16]. We consider an attacker capable of performing one precise fault injection out of 3 categories on a coefficient during data processing or storage. The attacker cannot affect the input parameters or the instruction processing part. We also consider that this attacker cannot perform a fault during the detection/correction process. We leave fault propagation for future work.

The three categories of faults against which we claim resilience are as follows:

- *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 computes a random datum. However, we reduce our claim regarding 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 randomised by this type of faults.
- *Zeroisation fault*: an attacker sets a code word to 0.

Our resilience to these faults is a direct consequence of the use of Reed-Solomon error-correcting codes. The detection of faults 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 that some faults have occurred. This detection can be performed with complexity $\mathcal{O}(l \log(l))$, and we can detect and correct up to $l-k$ faults.

4 Application example: ML-KEM

In this section, we propose to apply our work to ML-KEM as an example of application. Therefore, the value q in this section is $q = 3329$. An overview of the different sensitivities of sub-functions of ML-KEM is provided in Figure 11 for the ML-KEM Decapsulation, for PKE Encrypt in Figure 9 for PKE Decrypt in Figure 10. We use the following 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
- : Input/output of the algorithm with context-dependent sensitivity
- : Intermediate data with context-dependent sensitivity

Remark 8. : Non-sensitive hash function added to reduce the cost of the subsequent sensitive hash. This takes inspiration from several comments made during the NIST calls for comments on the draft of ML-KEM. Although it modifies the primitive, it only impacts the output in the case of a failure in the comparison of the ciphertexts. This modified ML-KEM thus has the same output as the standard in the case of a successful decapsulation and is indistinguishable from the standard one in the case of a failure while retaining the IND-CCA1 security from the original.

Remark 9. Our sensitivity schemes also have some minor differences from the algorithms described in Section 2.4. Notably, we group the use of $Decompress_q$ on the ciphertext into a single block named *Decode* in Figure 11, although its use has been described as part of the PKE Decrypt sub-function. The PKE Encrypt sensitivity scheme in Figure 9 does not include $Compress_q(\vec{u}, d_u)$ nor $Compress_q(v, d_v)$ as we use a different approach to the ciphertext comparison than the standard.

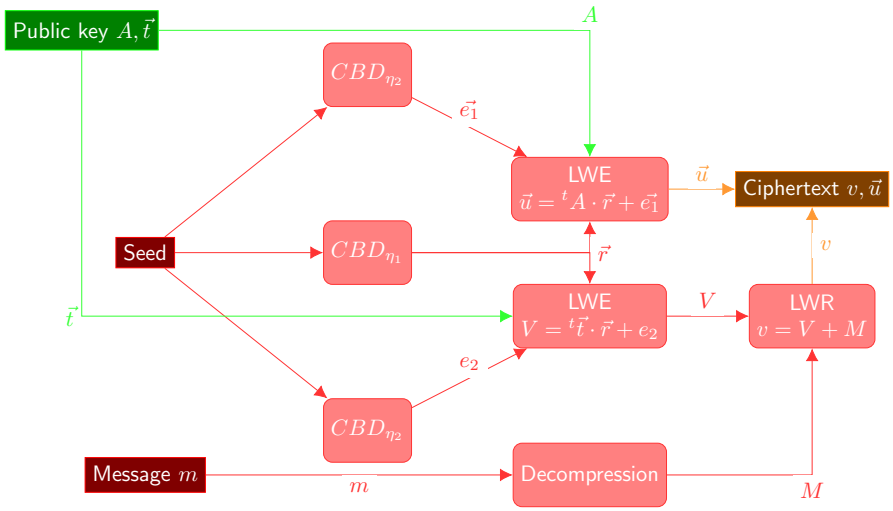


Figure 9: Sensitivity of the different sub-functions used in PKE Encrypt

Remark 10. The ciphertext $v, \vec{u}$ is sensitive if PKE Encrypt is used in the decapsulation, and is in this case denoted $v', \vec{u'}$ as in Figure 11 to distinguish the re-generated ciphertext from the received one. However, if PKE Encrpyt is used in the encapsulation, the ciphertext is public and denoted $v, \vec{u}$.

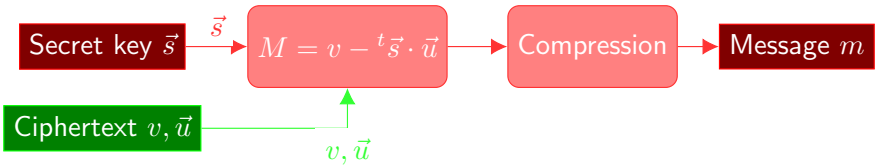


Figure 10: Sensitivity of the different sub-functions used in PKE Decrypt

4.1 Reed-Solomon Codes For ML-KEM

We list the possible Reed-Solomon encoders for several masking orders and each characteristic of ML-KEM in Table 2. We choose the length of the encoders to ensure that we can easily perform the `DFT`. This length must then be a factor of $q - 1 = 3328 = 13 * 2^8$ or $2^b - 1$ with $b \in \mathbb{N}^*$.

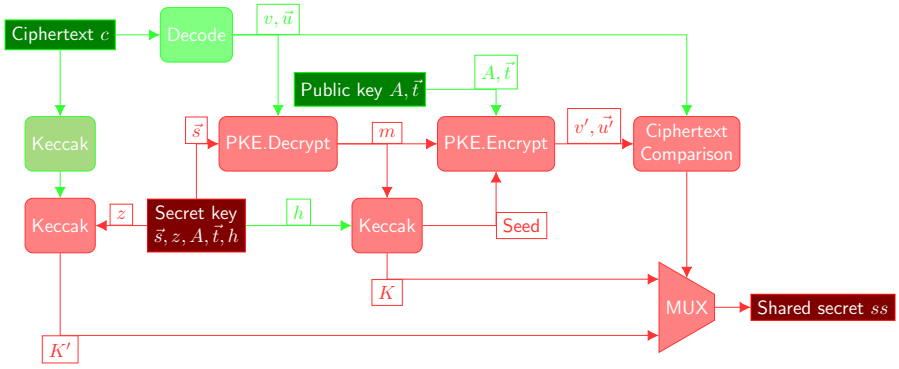


Figure 11: Sensitivity of the different sub-functions used in ML-KEM Decaps

Table 2: Possible encoders for different fields and masking orders up to order 6

Order	1	2	3	4	5	6
Over $\mathbb{F}_q$	$[4,2,3]$	$[8,3,6]$	$[8,4,5]$	$[13,5,9]$	$[13,6,8]$	$[13,7,7]$
Over $\mathbb{F}_{16}$	$[3,2,2]$	$[5,3,3]$	-	-	-	-
Over $\mathbb{F}_{64}$	$[3,2,2]$	$[7,3,5]$	$[7,4,4]$	$[9,5,5]$	-	-
Over $\mathbb{F}_{256}$	$[3,2,2]$	$[5,3,3]$	$[15,4,12]$	$[15,5,11]$	$[15,6,10]$	$[15,7,9]$

Remark 11. Reed-Solomon Code-Based Masking over $\mathbb{F}_q$ is applied to each polynomial coefficient in each vector or matrix object within ML-KEM. However, we can only mask one bit per code word when using RS-CBM on $\mathbb{F}_{2^b}$. This is due to the rotation operations within Keccak, which require us to manipulate bits separately.

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

$$A^{-1} \cdot V(\omega) = \begin{pmatrix} 14 & 10 & 6 \\ 15 & 11 & 7 \end{pmatrix},$$

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

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

We now focus on some non-linear parts of ML-KEM. Given the gadgets previously described in Section 3, linear functions within ML-KEM can be easily masked.

4.2 Message Compression and Ciphertext Comparison

State-of-the-art ML-KEM masked implementations [BC22, BGR+21] propose specific solutions to mask the non-linear parts of ML-KEM by using boolean formulae, where conversions are largely involved. In RS-CBM, these conversions are not yet cost-effective. To develop arithmetic 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.

The rationale behind these arithmetisations is to exploit the fact that the computations are performed on a finite field. This allows us to perform Lagrangian interpolations. We now discuss several algorithmic techniques to reduce the cost of evaluating such interpolations.

4.2.1 Paterson-Stockmeyer Fast Polynomial Evaluation in Characteristic $q \neq 2$

In ML-KEM, both the message compression and the ciphertext comparison act as "mappings" of values from $\mathbb{F}_q$ to $\{0,1\}$. Mathematically, this is equivalent to considering Lagrangian interpolations. Consequently, our methodology for securely computing these functions is based on the evaluation of polynomials. We use the work of Paterson and Stockmeyer [PS73] to minimise the number of sensitive multiplications required to evaluate these polynomials. Sensitive multiplications are more expensive than additions or scalar multiplications, and thus we reduce their number.

Remark 12. 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 let deg be its degree. First, assume 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:*

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

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

4.2.2 Message Compression

In the state-of-the-art masked implementations [BGR+21, BC22], the masked compression is performed using A2B conversions and Boolean formulae. As stated in Section 3.5, 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 0 or 1. If we perform a Lagrange interpolation, we have to evaluate a degree 3330 polynomial. However, we choose to evaluate only the part of the Lagrange interpolation mapping that points to 0. This results in a polynomial P_{FM} of degree 1665. By factoring the $(X-0)$ polynomial from P_{FM} and using the symmetric nature of the set of points assigned to 0, we evaluate a degree 832 polynomial in $Y = X^2$ and then multiply by the evaluation of X:

$$P_{FM}(X) = X \cdot \prod_{i=1}^{832} (Y - i^2). \tag{10}$$

The evaluation of Equation 10 performs the mapping to 0. However, the results that should be assigned to 1 are assigned 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 polynomial $P_F \in \mathbb{F}_q[X]$ performing the F mapping to 0;
Output: $F(\beta) \in \{0,1\}$;
Procedure `FPE`(β, P_F)**:**
  $\gamma = P_F(\beta)$;
  return γ^{q-1} *mod* q

Remark 13. 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 the ML-KEM case, we perform 13 sensitive multiplications after evaluating P_F.

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

4.2.3 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. Although in Algorithm 2 we propose 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 as Bos et al. [BGR+21] masked ML-KEM. Instead of compressing newly generated ciphertexts, we compare the decompression of the received ciphertext with 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 related to the parameters d_u and d_v. This is equivalent to checking that $\|\vec{u'} - \vec{u}\|_\infty \leq 2$ and $\|v' - v\|_\infty \leq 104$ in the case of ML-KEM-512.

Thus, a first step is to perform $\vec{u'} - \vec{u}$ and $v' - v$. As the ciphertext comparison should correspond to 0 or 1, we propose to check the inequalities for each polynomial coefficient separately by mapping them to 0 or 1. Then, we use the same method as Coron et al.[CGMZ23]: an arithmetised AND of an arithmetised NOT of each result is performed.

To check $\|U = (\vec{u'} - \vec{u})\|_\infty \leq 2$, we use a mapping denoted by FU such as if $U \in [\![-2, 2]\!]$, $FU(U) = 0$, otherwise $FU(U) = 1$. 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 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 $V \in [\![-104, 104]\!]$, $FV(V) = 0$, otherwise $FV(V) = 1$. 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. The mapping to 1 is performed once again according to Algorithm 2. An overview of the arithmetised ciphertext comparison is given in Algorithm 3.

Algorithm 3: Arithmetised ciphertext comparison [CGMZ23]

```
res = 1;
for each polynomial coefficient c in v − v' do
    res = res * (1−FPE(c, P_FV)) ;
end
for each polynomial coefficient in each component of u⃗ − u⃗' do
    res = res * (1−FPE(c, P_FU));
end
return res
```

4.2.4 Complexity

We use the methodology proposed in Algorithm 2 and our adaptation 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 and occurences within ML-KEM

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	$\times 256$

4.3 Constrained Conversions in ML-KEM

The state-of-the-art [BGR+21, BC22] relies heavily on unconstrained conversions to perform the non-linear functions of ML-KEM. Our conversions are constrained to sharings of binary values. Thus, we cannot repeat their designs. Although we arithmetised the computations of the message compression, using a fully arithmetised Keccak instance, while possible, is costly and inefficient. Thus, we use the advances of this paper on constrained conversions to be able to perform the Keccak with a Boolean RS-CBM.

The message compression is a function of `PKE.Decrypt` defined on $\mathbb{F}_q$ 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. It serves as an input to CBD instances in `PKE.Encrypt` during `KEM.Decaps`. Within these instances, bits are summed and subtracted according to Equation 4 in order to compute a random bounded value. Thus, ML-KEM requires that the conversion from $\mathbb{F}_q$ to $\mathbb{F}_{2^b}$ after compression in `PKE.Decrypt` and the inverse conversion are performed during the CBD instances.

Furthermore, to reduce the number of conversions performed, we propose to slightly modify Equation 4 by only adding bits together and subtracting the resulting bias afterwards:

$$CBD_\eta(\beta) = \sum_{i=0}^{n-1} \Big(\sum_{j=0}^{\eta} (\beta_{2i\eta+j} + \beta_{2i\eta+\eta+j}) - \eta \Big) X^i \text{ with } \beta \in \{0,1\}^{2\times n\times \eta} \tag{11}$$

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 4:

Table 4: 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

4.4 Cost-Amortisation in ML-KEM

We investigate the possible use of Cost-Amortisation within ML-KEM. A first application would be the parallel ciphering of several instances of ML-KEM. However, it is also possible to parallelise some parts of ML-KEM. We restrict ourselves to parallelising Code-Based Masking in characteristic 2 and leave the investigation of parallelisation of Code-Based Masking in characteristic q for future work, as the advances on switching between un-amortised and amortised RS-CBMs do not apply to characteristic q.

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

4.5 Comparison with Quadratic Masking and Recommendation

4.5.1 Security Issues with Quadratic Masking

In addition to not providing any FIA resilience, current state-of-the-art masked implementations of ML-KEM [BGR+21, BC22] do not resist Deep Learning SCA at low orders of masking [DNGW23, WBD24]. The resilience of RS-CBM to these types of attack has yet to be studied.

Also, arithmetic and Boolean maskings have limitations at higher orders, due to their quadratic complexity both in time and in randomness. Several works have also investigated 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.

4.5.2 Recommendation for ML-KEM

Reed-Solomon Code-Based Masking offers several advantages compared to quadratic masking due to its quasilinear complexity and its ability to correct errors. However, it has some drawbacks. In particular, it does not efficiently mask Keccak instances, as stated in Section 4.1. Our recommendation for a masked implementation of ML-KEM is to use our design on the "arithmetic" parts of the primitive and to use an externalised Keccak instance with specific countermeasures against SCA (e.g. Boolean quadratic masking) and against FIA (e.g. physical mesh or a repetition code with a masked majority vote).

Another remark about the application of RS-CBMs over $\mathbb{F}_q$ is which masking order to use. The performance of this type of masking mostly depends on the code length, which, according to Table 2, can be the same for several orders of masking. Thus, we recommend for ML-KEM the use of orders $1, 3$ and 6 when using RS-CBM over $\mathbb{F}_q$, as the differences in complexity between orders 2 and 3, and between orders $4, 5$ and 6, are minor.

5 Conclusion

In this paper, we propose new advances in Reed-Solomon Code-Based Masking and used a post-quantum KEM as an application example. We expanded on the work of Carlet et al. [CDGT24] by detailing its application to $\mathbb{F}_q$, and proposed constrained conversions between

arithmetic and Boolean RS-CBMs. We also proposed a methodology to switch between "amortised" and "un-amortised" RS-CBMs in $\mathbb{F}_2$ in the context of Cost-Amortisation (CA). Formal verifications were performed using the t-(S)NI criteria, as well as experimental verification following the TVLA methodology. The resilience of faults is also discussed.

ML-KEM is used as an example of an application of our advances. We propose the first fully arithmetised masking strategies for both message compression and ciphertext comparison. To do so, we rely on Lagrangian interpolation. Complexity optimisations are performed by applying Fermat's Little Theorem and using the Paterson-Stockmeyer fast polynomial evaluation to ensure optimal complexity when evaluating these interpolations. Constrained conversions and amortisation-switching is used to handle the Keccak instances within ML-KEM. Other applications of our advances on RS-CBM to ML-KEM, notably the use of constrained conversions allowing a less costly Keccak instance using a Boolean RS-CBM, and the use of Cost-Amortisation to algorithmically parallelise the computation of CBD instances, are also investigated.

Future works will revolve around proposing unconstrained and efficient conversions between Reed-Solomon Code-Based Maskings. A complete implementation of ML-KEM can be investigated using the hybrid design recommended in Section 4.5.2. Finally, as quadratic masking seems to not be sufficient against Deep Learning assisted Side-Channel opponents, a thorough study of Deep Learning SCA against RS-CBM is of interest.

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.

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. doi:10.6028/NIST.IR.8413-upd1.

[ABC+25] Gorjan Alagic, Maxime Bros, Pierre Ciadoux, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, et al. Status report on the fourth round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2025. doi:10.6028/NIST.IR.8545.

[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. doi:10.1145/2976749.2978427.

[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. doi:10.1109/FDTC.2016.11.

[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. doi:10.46586/tches.v2022.i4.553-588.

[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. doi:10.1007/978-3-662-53140-2_2.

[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. doi:10.1007/978-3-031-38548-3_13.

[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. doi:10.46586/tches.v2021.i4.173-214.

[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. doi:10.46586/tches.v2021.i3.202-234.

[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.

[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. doi:10.46586/tches.v2024.i1.398-432.

[CGMZ23] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order polynomial comparison and masking lattice-based encryption. *IACR TCHES*, 2023(1):153–192, 2023. doi:10.46586/tches.v2023.i1.153-192.

[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. doi:10.1007/3-540-48405-1_26.

[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. doi:10.1007/978-3-662-43933-3_21.

[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. doi:10.1145/3591866.3593072.

[Dwo15] Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. *NIST FIPS*, 2015. doi:10.6028/NIST.FIPS.202.

[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. doi:10.1007/978-3-319-75208-2_7.

[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. doi:10.1007/3-540-48405-1_34.

[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.

[GJR18] Dahmun Goudarzi, Antoine Joux, and Matthieu Rivain. How to securely compute with noisy leakage in quasilinear complexity. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 547–574. Springer, Heidelberg, December 2018. doi:10.1007/978-3-030-03329-3_19.

[GPRV21] Dahmun Goudarzi, Thomas Prest, Matthieu Rivain, and Damien Vergnaud. Probing security through input-output separation and revisited quasilinear masking. *IACR TCHES*, 2021(3):599–640, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8987. doi:10.46586/tches.v2021.i3.599-640.

[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. doi:10.1109/TC.2022.3197073.

[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. doi:10.1007/978-3-540-45146-4_27.

[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. doi:10.46586/tches.v2024.i2.844-869.

[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. doi:10.1007/3-540-68697-5_9.

[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. doi:10.1109/TVLSI.2012.2231707.

[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. doi:10.1007/978-3-642-13190-5_1.

[MLD24] Module-lattice-based digital signature. National Institute of Standards and Technology, NIST FIPS PUB 204, U.S. Department of Commerce, August 2024. doi:10.6028/NIST.FIPS.204.

[MLK24] Module-lattice-based key-encapsulation mechanism. National Institute of Standards and Technology, U.S. Department of Commerce, August 2024. URL: url{https://doi.org/10.6028/NIST.FIPS.203}, doi:10.6028/NIST.FIPS.203.

[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. doi:10.13154/tches.v2018.i1.142-174.

[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. doi:10.1137/0202007.

[RS02] Eric M Rains and Neil JA Sloane. Self-dual codes. *arXiv preprint math/0208001*, 2002. doi:10.48550/arXiv.math/0208001.

[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. doi:10.1109/SFCS.1994.365700.

[SLH24] Stateless-hash-based digital signature algorithm. National Institute of Standards and Technology, NIST FIPS PUB 203, U.S. Department of Commerce, August 2024. doi:10.6028/NIST.FIPS.205.

[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. doi:10.1007/978-3-031-54776-8_12.

[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. doi:10.13154/tches.v2020.i2.128-171.

[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. doi:10.1109/49.1926.