IACR Communications in Cryptology
Vol. 0, No. 0, 27 pages.

Optimizing Key Recovery in *Classic McEliece*: Advanced Error Correction for Noisy Side-Channel Measurements

Nicolas Vallet[1], Pierre-Louis Cayrel[1], Brice Colombier[1], Vlad-Florin Drăgoi[2,3] and Vincent Grosso[1]

[1] Université Jean Monnet Saint-Etienne, CNRS, Institut d Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, Saint-Étienne, France
[2] Faculty of Exact Sciences, Aurel Vlaicu University of Arad, Arad, Romania
[3] LITIS, University of Rouen Normandie, Saint-Étienne du Rouvray, France

Abstract. *Classic McEliece* was one of the code-based Key Encapsulation Mechanism finalists in the NIST post-quantum cryptography standardization process. Several key-recovery side-channel attacks on the decapsulation algorithm have already been published. However none of them discusses the feasibility and/or efficiency of the attack in the case of noisy side-channel acquisitions. In this paper, we address this issue by proposing two improvements on the recent key-recovery attack published by Drăgoi et al.. First, we introduce an error correction algorithm for the lists of Hamming weights obtained by side-channel measurements, based on the assumption, validated experimentally, that the error on a recovered Hamming weight is bounded to ± 1. We then offer a comparison between two decoding efficiency metrics, the theoretical minimal error correction capability and an empirical average correction probability. We show that the minimal error correction capability, widely used for linear codes, is not suitable for the (non-linear) code formed by the lists of Hamming weights. Conversely, experimental results show that out of 1 million random erroneous lists of $2t = 128$ Hamming weights, only 2 could not be corrected by the proposed algorithm. This shows that the probability of successfully decoding a list of erroneous Hamming weights is very high, regardless of the error weight. In addition to this algorithm, we describe how the secret Goppa polynomial g, recovered during the first step of the attack, can be exploited to reduce both the time and space complexity of recovering the secret permuted support $\mathcal{L}$.

Keywords: Post-quantum cryptography · Code-based cryptography · Classic McEliece · Side-channel attacks

1 Introduction

Since 2017, when the NIST call for PQC proposals was issued, a great deal of effort has been expended in developing new post-quantum standards. In August 2024, four algorithms have been selected:

- CRYSTALS-KYBER a solution based on structured lattices became the FIPS 203, knwon as ML-KEM. [oST24a]

E-mail: nicolas.vallet@univ-st-etienne.fr (Nicolas Vallet), pierre.louis.cayrel@univ-st-etienne.fr (Pierre-Louis Cayrel), b.colombier@univ-st-etienne.fr (Brice Colombier), vlad.dragoi@uav.ro (Vlad-Florin Drăgoi), vincent.grosso@univ-st-etienne.fr (Vincent Grosso)

This work is licensed under a "CC BY 4.0" license.

- CRYSTALS-Dilithium, a digital signature based on structured lattices became FIPS 204 - ML-DSA. [oST24b]
- SPHINCS+, a conservative solution for digital signatures based on hash function became FIPS 205 - SLH-DSA. [oST24c]
- Falcon, a digital signature built on structured lattices will become FN-DSA. [PFH+22]

NIST wanted to have cryptosystems whose security is based on different hard problems. The aim is to extend the study of other algorithms which are based on hard problems from competing theories, error-correcting codes in this case. As such, three code-based candidates were proposed for a fourth round of investigation: BIKE [ABB+22], *Classic McEliece* [ABC+22] and HQC [AAB+22]. In March 2025 HQC was selected. Due to the common features between BIKE and HQC, the two teams decided to merge and form a single and strong group for HQC. *Classic McEliece* was not selected at this stage, as it is currently under review by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). Despite its exclusion from the NIST competition, the scheme remains of significant interest due to its ongoing evaluation by these international standardization bodies. Indeed, a joint technical committee was created by these two organizations. As pointed out ISO/IEC expects to standardise the key encapsulation mechanisms FrodoKEM [NAB+20] and *Classic McEliece* [ABC+22]. We notice that a significant factor in the ISO/IEC decision was the security of the two aforementioned solutions, FrodoKEM and *Classic McEliece* being more conservative than their structured lattice-based analogue ML-KEM.

At this point in the process, particular attention is paid to the practical aspects, especially implementation issues and vulnerability to physical attacks.

1.1 Side-channel attacks on the *Classic McEliece* KEM

Like all KEM cryptosystems, *Classic McEliece* is composed of three main algorithms:

KeyGen (detailed description in Algorithm 3): the key generation algorithm that selects a random permuted support $\mathcal{L}$ composed of n elements of $\mathbb{F}_{2^m}$ and an irreducible monic polynomial g of degree t. The pair $(g, \mathcal{L})$ forms the secret key sk that will be used to compute the private parity-check matrix $\boldsymbol{H}_{\text{priv}}$. Then, $\boldsymbol{H}_{\text{priv}}$ is reduced into its systematic form $\boldsymbol{H}_{\text{pub}} = (\boldsymbol{I}_{\text{mt}}|\mathsf{pk})$.

Encap (detailed description in Algorithm 4): a random error vector $\boldsymbol{e} \in \mathbb{F}_2^n$ of Hamming weight t is generated and the ciphertext is computed as the product $\boldsymbol{ct} = \boldsymbol{H}_{\text{pub}}\boldsymbol{e}^T$. The session key K is computed by hashing $\boldsymbol{e}$ and $\boldsymbol{ct}$.

Decap (detailed description in Algorithm 5): $\boldsymbol{e}$ is recovered from the ciphertext $\boldsymbol{ct}$ with the knowledge of g and $\mathcal{L}$. The session key K is then computed by hashing $\boldsymbol{e}$ and $\boldsymbol{ct}$.

An attack on the cryptosystem has the potential to recover the long-term secret, which is represented by the variable $\boldsymbol{H}_{\text{priv}}$ in the KeyGen and Decap algorithms. Such an attack can be used to eavesdrop on any conversation that occurs either before or after the key recovery.

Several key-recovery side-channel attacks on the NIST reference implementation of *Classic McEliece* have been presented with different ways to handle the classifier accuracy in the attack and different leakage models.

Brinkmann et al. [BCM+25] describe a full key-recovery attack on the key generation algorithm. More preciselly, they attack the Gaussian elimination step which allows them to recover the columns of the private key $\boldsymbol{H}_{\text{priv}}$.

Seck et al. [SCD+23] proposed a partial key-recovery attack on the Decapsulation algorithm where they recover the Goppa polynomial g by exploiting the Hamming weight of its coefficients.

Guo et al. [GJJ22] present a full key-recovery attack on the decapsulation algorithm. Their solution requires particular chosen plaintexts.

Colombier et al. [DCV+25] propose a full key-recovery attack on the syndrome computation step in the decapsulation algorithm.

In Table 1 we summarise all attacks targeting the decapsulation step and the attacks scenarios, as well as our contribution.

Table 1: Comparison of side-channel key-recovery attacks on the Decapsulation step in the *Classic McEliece* KEM for **noisy scenarios**. The ChipWhisperer hardware setup [OC14] was used for all articles.

Feature	[GJJ22]	[SCD+23]	[DCV+25]	**This article**
Target op.	FFT $\sigma(x)$	$\mathsf{load}(g)$	Syndrome comput.	Syndrome comput.
Noise	–	–	–	Hamming weight ± 1
Accuracy	100 %	100 %	> 94.5 %	> 81 %[a]
Implem.	Ref. & optim.	Optimized	Reference	Reference
Hardware	FPGA & ARM	ARM	ARM	ARM

[a] The accuracy in this article is computed in the ± 1 error model.

1.2 Contributions

In this paper we improve the attack proposed in [DCV+25] on several aspects presented below. Algorithm 1 highlights the changes done for the proposed improved attack in red.

Algorithm 1 Overview of the improved attack presented in this article.

Input: A noisy side-channel trace of the execution of the *Classic McEliece* Decapsulation
Output: The private key $\mathsf{sk} = (g, \mathcal{L})$

1: Estimate the Hamming weight of $\left(\alpha^i g^{-2}(\alpha)\right)_{i=0}^{2t-1} \forall \alpha \in [\![0; mt-1]\!]$
2: Correct the erroneous lists of Hamming weights ▷ Subsection 4.3
3: Recover t pairs $(\alpha, g(\alpha))$
4: Recover polynomial g from t pairs $(\alpha, g(\alpha))$ via interpolation
5: Recover the first mt pairs $(\alpha, g(\alpha))$ from a reduced Hamming weight Vandermonde distinguisher ▷ Subsection 5.1
6: Construct $\boldsymbol{H}_{\mathrm{priv}_g}[:, [\![0; mt-1]\!]]$ using g and the first mt αs
7: Recover $\boldsymbol{H}_{\mathrm{priv}_g} = \boldsymbol{H}_{\mathrm{priv}_g}[:, [\![0; mt-1]\!]]\boldsymbol{H}_{\mathrm{pub}}$ ▷ Subsection 5.2
8: Recover the full permuted support $\mathcal{L} = \frac{\boldsymbol{H}_{\mathrm{priv}_g}[1,:]}{\boldsymbol{H}_{\mathrm{priv}_g}[0,:]} \left(= \frac{\alpha_j g^{-1}(\alpha_j)}{g^{-1}(\alpha_j)}\right)$

Error correction on the estimated Hamming weight lists using the Hamming weight Vandermonde distinguisher Once the Hamming weights have been estimated in [DCV+25], no error correction is made on the, potentially wrong, lists of Hamming weights. Line 2 of Algorithm 1, uses the Hamming weight Vandermonde-like matrix $\beta \boldsymbol{V}_\alpha = \left(\mathsf{wt}(\alpha^i \beta)\right)_{i \in [\![0;2t]\!]}$ for $(\alpha, \beta \in \mathbb{F}_{2^m}^* \times \mathbb{F}_{2^m}^*)$, with $\mathsf{wt}(\alpha)$ the Hamming weight of the element $\alpha \in \mathbb{F}_{2^m}$, defined in [DCV+25], and applies a new error correction algorithm on the estimated Hamming weights. The efficiency of this new error correction algorithm is then analysed.

Reduction of the required number of the Vandermonde matrix columns using the recovered Goppa polynomial In [DCV+25], due to the lack of error correction in the estimated Hamming weights as well as the probability of collisions between the lists of Hamming weights of several pairs $(\alpha, g(\alpha))$, the authors need more than mt columns of the Vandermonde matrix to recover the permuted support $\mathcal{L}$ with a high probability. As only the product of the ciphertext $\boldsymbol{ct}$ with the mt first columns of the Vandermonde matrix is required for the syndrome computation, the authors propose to shorten the Vandermonde matrix as a countermeasure to the attack. We show in line 5 of Algorithm 2 how the Goppa polynomial g, recovered with t pairs $(\alpha, g(\alpha))$, can be exploited to reduce the size of the Hamming weight Vandermonde-like matrix $\beta \boldsymbol{V}_\alpha$. This size reduction allows to recover the pairs $(\alpha, g(\alpha))$ corresponding to the first mt columns of the Vandermonde matrix with high probability, therefore, making the countermeasure useless.

Organization This article is organized as follows: In Section 2, we introduce the notations and the necessary background in coding theory and code-based cryptography. We also introduce template attacks and a detailed presentation of the *Classic McEliece* KEM. In Section 3, we present the existing side-channel key recovery attacks on the *Classic McEliece* KEM and the way they deal with their classifier's accuracy. In Section 4, we first define the attacker model and the error model. We then present our error correction algorithm and finally propose different metrics to characterize its correcting efficiency. In Section 5, we describe how the Goppa polynomial g recovered during the attack can be exploited to improve the recovery of the permuted support $\mathcal{L}$. In Section 6, we show the experimental results. Finally, in Section 7, we conclude this paper and discuss possible future works.

Reproducibility To allow the reproducibility of our results, the source codes of the proposed attack is available on the following GitLab repository: `https://gitlab.univ-st-etienne.fr/sesam/cic-2025-2-optimizing-key-recovery-in-classic-mceliece-advanced-error-correction-for-noisy-side-channel-measurements`

2 Background

2.1 Linear algebra, norms and distances

Some of the notations used in this article are extracted from the design document of round-4 *Classic McEliece* [ABC+22].

Notations Lowercase letters are used to denote integers and intervals of integers are denoted by $[\![a; b]\!]$. Sets are denoted using caligraphic capital letters, e.g., $\mathcal{A}$, and by $\#\mathcal{A}$ we denote the cardinality of the set $\mathcal{A}$. An algebraic structure will be denoted by $\mathbb{K}$, in particular $\mathbb{K} = \mathbb{Z}$ denotes the ring of integers and $\mathbb{K} = \mathbb{F}_q$ denotes the finite field of order q. Elements of the finite field $\mathbb{F}_q$ are represented with lower Greek letters. Bold lowercase letters are used to denote vectors, e.g. $\boldsymbol{v}$, while bold uppercase letters are used for matrices.

Let $\boldsymbol{v}$ be a vector in row notation, its ith coordinate is denoted by v_i and the transpose of the vector is denoted by $\boldsymbol{v}^T$. Let $\boldsymbol{H}$ be a matrix, we denote by h_{ij} the coefficient of $\boldsymbol{H}$ located at the ith row and j^{th} column. We denote a row or a column of $\boldsymbol{H}$ by $\boldsymbol{H}[i, :]$ and $\boldsymbol{H}[:, j]$ respectively. Also, a sub-matrix of $\boldsymbol{H}$ indexed by a set of rows $\mathcal{I}$ and a set of columns $\mathcal{J}$ is denoted by $\boldsymbol{H}[\mathcal{I}, \mathcal{J}]$.

Here, we will deal with objects defined over $\mathbb{F}_{2^m}$, which is constructed as an extension of $\mathbb{F}_2$ using an irreducible polynomial $f(x) \in \mathbb{F}_2[x]$, with $\deg(f) = m$, defined in the specification of *Classic McEliece* [ABC+22], such that $\mathbb{F}_{2^m} = \mathbb{F}_2/f(x) = \mathbb{F}_2(\lambda)$ where

λ is the root of $f(x)$. Also, one can rewrite $\alpha = \boldsymbol{\alpha} \cdot (\lambda^0, \lambda^1, \ldots, \lambda^{m-1})^T$ where $\boldsymbol{\alpha} = (\alpha_0, \ldots, \alpha_{m-1}) \in \mathbb{F}_2^m$. We call $\boldsymbol{\alpha}$ the binary extension of α.

Definition 1 (**Hamming weight**)**.** Let $\boldsymbol{c} \in \mathbb{K}^n$. Its Hamming weight is $\mathsf{wt}(\boldsymbol{c}) = \#\{i \in [\![0; n-1]\!] \mid c_i \neq 0\}$.

For $\alpha \in \mathbb{F}_{2^m}$ we define its Hamming weight by $\mathsf{wt}(\alpha) \stackrel{\mathsf{def}}{=} \mathsf{wt}(\boldsymbol{\alpha})$, where $\boldsymbol{\alpha} \in (\mathbb{F}_2)^m$ is the binary extension of α.

Notice that for any $\boldsymbol{c} \in \mathbb{K}^n$ we have $\mathsf{wt}(\boldsymbol{c}) \in \mathbb{Z}$, more exactly $\mathsf{wt}(\boldsymbol{c}) \in [\![0; n]\!]$, while $\mathsf{wt}(\alpha) \in [\![0; m]\!]$ for any $\alpha \in \mathbb{F}_{2^m}$.

Definition 2 (**Hamming distance**)**.** Let $(\boldsymbol{c}, \boldsymbol{c}') \in (\mathbb{K}^n)^2$. The Hamming distance between $\boldsymbol{c}$ and $\boldsymbol{c}'$ is defined as $d_0(\boldsymbol{c}, \boldsymbol{c}') = \mathsf{wt}(\boldsymbol{c} - \boldsymbol{c}')$.
For $(\alpha, \beta) \in (\mathbb{F}_{2^m})^2$ we define the Hamming distance between α and β by $d_0(\alpha, \beta) \stackrel{\mathsf{def}}{=} d_0(\boldsymbol{\alpha}, \boldsymbol{\beta})$.

Definition 3 (**Infinite distance**)**.** Let $(\boldsymbol{c}, \boldsymbol{c}') \in (\mathbb{Z}^n)^2$. The infinite distance between $\boldsymbol{c}$ and $\boldsymbol{c}'$ is defined as $d_\infty(\boldsymbol{c}, \boldsymbol{c}') = \max_{i \in [\![0; n-1]\!]} |\boldsymbol{c}_i - \boldsymbol{c}'_i|$.

In this article, we deal with objects called lists of Hamming weights, which are vectors of the form $\boldsymbol{x} = (\mathsf{wt}(\alpha_1), \ldots, \mathsf{wt}(\alpha_n)) \in \mathbb{Z}^n$ where for all $i \in [\![1; n]\!], \alpha_i \in \mathbb{F}_{2^m}$. These should not be confused with Hamming weight of vectors from the extension field, i.e., $\mathsf{wt}(\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}$.

2.2 Code-based cryptography and the *Classic McEliece* KEM

Classic McEliece is a code-based cryptosystem that uses a Goppa code. The following section presents the fundamental definitions that are essential for an understanding of the *Classic McEliece* algorithms.

Definition 4 (**Code**)**.** A code $\mathscr{C}$ of length n is a subset of $\mathbb{F}_q^n$. An element $\boldsymbol{c} = (c_0, \ldots, c_{n-1}) \in \mathscr{C}$ is called a codeword. If $\mathscr{C}$ is a vector subspace of dimensions k of the vector space $\mathbb{F}_q^n$, then $\mathscr{C}$ is a linear code of length n and dimension k. Such a linear code is called an $[n, k]_q$ code.

Definition 5 (**Parity-check matrix**)**.** Let $\mathscr{C}$ be a $[n, k]_q$ linear code and $\boldsymbol{H}$ be an $(n-k) \times n$ matrix such that

$$\forall \boldsymbol{c} \in \mathscr{C} \Leftrightarrow \boldsymbol{H}\boldsymbol{c}^T = 0.$$

Then $\boldsymbol{H}$ is a parity-check matrix of $\mathscr{C}$.
Let $\boldsymbol{v} \in \mathbb{F}_q^n$, then the syndrome of the vector $\boldsymbol{v}$ with respect to $\boldsymbol{H}$ is defined as the vector $\boldsymbol{s} = \boldsymbol{H}\boldsymbol{v}^T$.

Definition 6 (**Goppa Code**)**.** Let $\mathcal{L} = \{\alpha_0, \ldots, \alpha_{n-1}\}$ with $\alpha_i \in \mathbb{F}_{2^m}$ and $\alpha_i \neq \alpha_j$ for $i \neq j$. Let $g(x) \in \mathbb{F}_q[x], \deg(g) = t$ such that $\forall \alpha \in \mathcal{L}, g(\alpha) \neq 0$. Then the set

$$\mathscr{G}(g, \mathcal{L}) = \left\{ \boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_2^n : \sum_{i=0}^{n-1} \frac{c_i}{x - \alpha_i} \equiv 0 \mod g(x) \right\}$$

is a Goppa code with parameters $\mathcal{L}$ and $g(x)$.
Also, let $\boldsymbol{H}$ be the $t \times n$ matrix defined as

$$\boldsymbol{H} = \begin{pmatrix} g^{-1}(\alpha_0) & \ldots & g^{-1}(\alpha_{n-1}) \\ \vdots & \ddots & \vdots \\ \alpha_0^{t-1} g^{-1}(\alpha_0) & \ldots & \alpha_{n-1}^{t-1} g^{-1}(\alpha_{n-1}) \end{pmatrix}.$$

Then, its expansion over $\mathbb{F}_2$, which is an $mt \times n$ matrix, is a parity-check matrix of $\mathscr{G}(g, \mathcal{L})$.

Classic McEliece *Classic McEliece* [ABC+22] is a KEM relying on the Niederreter cryptosystem based on a Goppa code. Table 2 shows the sets of parameters for *Classic McEliece* as well as a smaller set of parameters, called *toyeliece* [BCM+25], that is a toy KEM useful for research purposes. We give a description of the KEM algorithms in Appendix A.

Table 2: Parameter sets for `toyeliece` and *Classic McEliece*.

	toyeliece51220	mceliece348864	mceliece460896
(m, n, t)	(9, 512, 20)	(12, 3488, 64)	(13, 4608, 96)
	mceliece6688128	mceliece6960119	mceliece8192128
(m, n, t)	(13, 6688, 128)	(13, 6960, 119)	(13, 8192, 128)

2.3 Template attacks

Introduced in 2003 by Chari et al. [CRR03] as "*the strongest form of side-channel attack possible in an information theoretic sense*", template attacks have become a popular type of side-channel attack. They rely on the assumption that the relation between acquired power consumption traces and secret data can be modeled as multivariate Gaussian distributions, called templates, defined by a mean vector and a covariance matrix.

During the first stage of the attack, called the profiling stage, the attacker uses an identical copy of the attacked device on which they have full control to run the target cryptographic algorithm many times with different values for the secret parameter while recording the power consumption. The power consumption traces are then used to estimate the parameters (mean and covariance) of the corresponding multivariate Gaussian distributions.

During the second stage, called the matching stage, the attacker records power consumption traces from the attacked device and then uses the templates built previously to associate each trace with its corresponding secret value.

3 Related works

3.1 Key-recovery side-channel attacks on *Classic McEliece*

We focus on profiled key-recovery side-channel attacks on *Classic McEliece* and how they handle the classifier's inaccuracy. In particular message-recovery attacks, as well as key-recovery attacks not relying on side-channel, are out of scope.

In [BCM+25], a key-recovery attack on the key generation algorithm is proposed. They show how power consumption leakage during the Gaussian elimination step, where an invertible matrix $\boldsymbol{E}$ is used to transform the binary matrix $\boldsymbol{H}'$ into the matrix $\boldsymbol{H}_{\text{pub}}$, can be used to recover the matrix $\boldsymbol{E}$.

In [GJJ22], the authors proposed a chosen-ciphertext profiled attack on the decapsulation algorithm. They exploit a vulnerability in the additive FFT evaluation of the error locator polynomial. For a given error-locator polynomial, the FFT behaves the same. Therefore, side-channel analysis can be used to recover the error locator polynomial and the secret support. To do so, they train a neural network as a classifier to associate the power consumption during the FFT evaluation of an error locator polynomial of degree 1 (i.e. $\sigma(x) = (x - \alpha_i)$) with the corresponding α_i. Since there are 2^m α_i, there are 2^m possibilities for the degree 1 error locator polynomial.

In [SCD+23], the proposed attack is also a profiled one on the decapsulation algorithm but the ciphertexts used are random. In this article, the authors exploit a vulnerability during the loading of the Goppa polynomial g which allows to recover the Hamming weight

of the coefficients of the polynomial. They then find g by searching in the set of all monic polynomial of degree t with corresponding coefficients Hamming weights until they find a polynomial such that the Goppa code $\mathscr{G}(g, \mathcal{L})$ is equivalent to the public code $\mathcal{C}$. Finally, they use the Support Splitting Algorithm [Sen00] to find the permutation $\mathcal{L}$ of the secret key.

In [DCV+25], the authors described a profiled attack using random ciphertexts on the decoding algorithm. They exploit a vulnerability during the syndrome computation which allows them to recover the Hamming weights of $\left(\alpha^i g^{-2}(\alpha)\right)_{i=0}^{2t-1}$ $\forall \alpha \in \mathcal{L}$. They then show how those lists of Hamming weights can be exploited to recover the corresponding pair $(\alpha, g(\alpha))$. Once they recover t pairs among the n pairs, they use Lagrange interpolation to recover the Goppa polynomial g. Finally, they use mt recovered pairs $(\alpha, g(\alpha))$, such that the corresponding columns of the matrix $\boldsymbol{H}'$ are linearly independent, to recover the parity-check matrix $\boldsymbol{H}$, and therefore the rest of the permuted support $\mathcal{L}$, from the public key $\mathsf{pk} = \boldsymbol{T}$.

3.2 Dealing with inaccurate side-channel classifiers

In [BCM+25], the attack uses an algorithm that can correct bit-flip errors up to $\tau \approx 0.4$ in the recovered matrix $\boldsymbol{E}$ used in the Gaussian elimination step.

In [GJJ22], [SCD+23] and [DCV+25], the power consumption traces are acquired using a ChipWhisperer [OC14], allowing a high accuracy for the classifier (100% for [GJJ22], 99.86% for [SCD+23] and 96.84% for [DCV+25] in the worst case).

Due to its high classifier accuracy, no considerations are made in [SCD+23] about the impact of the accuracy on the feasibility of the attack. In particular, no information is given on the complexity to recover the Goppa polynomial if the Hamming weights of the Goppa polynomial coefficients are wrongly recovered.

In [GJJ22], the authors consider the case of a *threshold approach* where instead of choosing the α with the maximum likelihood given by their deep-learning model, they pick a threshold τ such that if the largest likelihood value is still below that threshold, then they measure new power consumption traces of the decryption process using the same error vector $\mathbf{e}_i$ until the maximum likelihood outputted by the classifier is above the threshold. However, no information is given about this threshold value or the number of new power consumption traces needed with regard to the efficiency of the deep-learning classifier. Moreover, no information is given on how the attack will perform if the classifier makes a wrong guess on one, or more, α.

In [DCV+25], the authors shows how the accuracy affects the success probability of the attack. Moreover, they suggest an algorithm proposed by Bernstein [Ber24] that could be used to interpolate the polynomial g even if some recovered pairs $(\alpha, g(\alpha))$ are wrong. In addition, since they need only $mt + \delta$ pairs, out of the n used in the syndrome computation, to recover the permuted support $\mathcal{L}$, they propose to reduce the length of the list of Hamming weight from $2t$ to $d_{m,t} < 2t - 1$ chosen such that at least $mt + \delta$ pairs can be recovered. This reduction allows to reduce the probability to have an erroneous Hamming weight in the list.

4 Error correction algorithm

We first introduce the attacker model followed by the error model. Then, we describe our error correction algorithm and finally, we present the error correction capability and the correction probability, two metrics for analyzing the efficiency of our error correction algorithm.

4.1 Attacker model

Our main purpose is to improve the attack proposed in [DCV+25] by decreasing the minimal accuracy required by the classifier as well as the number of pairs $(\alpha,\ g(\alpha))$ required. Hence, our attacker model will be almost identical to the one in [DCV+25] with an improvement on the required accuracy. Indeed, while their attacker model requires the Hamming weights outputted by the classifier to be correct with a probability of 0.94[1] with unbounded error, we only require the Hamming weights outputted by our classifier to be in the ± 1 interval around the correct value, thus bounded error. We show in Subsection 4.2 that this bounded error model holds for an accuracy higher than 0.81.

The attacker model that we use here was previously employed by [SCD+23], and contrary to [GJJ22], *no control* over the ciphertext is required. It is based on the following requirements.

1. A clone device running the reference implementation of *Classic McEliece* is used by the attacker with a complete control of the inputs while measuring the power consumption.

2. Only one power trace of the syndrome computation of the reference implementation of the *Classic McEliece* decapsulation algorithm has been recorded by the attacker.

3. There is no control nor knowledge by the attacker of the input ciphertext used on the attacked device.

4.2 Error model

We assume that an attacker can recover the Hamming weight of an intermediate value, for example thanks to side-channel measurement of the data. However, the guessed values can be incorrect, but the guessed Hamming weights are "close" to their actual value. By close we mean that we recover the Hamming weight or the Hamming weight plus one or the Hamming weight minus one.

In other words, let us denote by α an intermediate value and by $\mathsf{wt}(\alpha)$ its Hamming weight, the predicted Hamming weight $\hat{\mathsf{wt}}(\alpha)$, then we have $-1 \leq \mathsf{wt}(\alpha) - \hat{\mathsf{wt}}(\alpha) \leq 1$. Both $\mathsf{wt}(\alpha)$ and $\hat{\mathsf{wt}}(\alpha)$ are integers thus the error is in $\{-1, 0, 1\}$. For short we refer to this assumption as the ± 1 error model. Notice that if $\mathsf{wt}(\alpha) = m$, the case where $\hat{\mathsf{wt}}(\alpha) = m+1$ is not possible in our model. In the same way, if $\mathsf{wt}(\alpha) = 0$, the case where $\hat{\mathsf{wt}}(\alpha) = -1$ is not possible.

To find the accuracy where this error model is valid, we use the most common leakage model, the Hamming weight plus Gaussian leakage model [CRR03, MMPS09, Riv09, HRG14]. We consider a Gaussian leakage function with $\mathcal{V}_\alpha \sim \mathcal{N}(\mu_\alpha, \sigma^2)$, where $\alpha \in \mathbb{F}_{2^m}$, $\mathcal{V}$ is the leakage view, $\mathcal{N}$ is the normal distribution with mean $\mu_\alpha = c \times \mathsf{wt}(\alpha) + \mu_0$, with c a constant and μ_0 the mean for $\alpha = 0 \in \mathbb{F}_{2^m}$ and standard deviation σ. Choudary showed that the parameter σ is the same for all α and we can assume $\mu_0 = 0$ [Cho14]. This model leakage allows to have the formula for the accuracy presented in Equation 1.

$$\mathsf{accuracy}(c, \sigma) = \mathsf{erf}\left(\frac{c}{2\sigma\sqrt{2}}\right) \tag{1}$$

where erf is the Gauss error function. It is important to note that the accuracy depends on $\frac{c}{\sigma}$, therefore only that ratio is important rather than specific values for c and σ. In this article, we will use the accuracy, deduced experimentally or estimated during simulations, to compute the efficiency of our error correction algorithm, so the actual values of c and σ

[1] Such high accuracy can be obtained with low-noise dedicated platforms, such as the ChipWhisperer [OC14], but may be harder to reach for real-life devices.

are not relevant for the algorithm performance. We can then define the probability to have an error outside the ± 1 error model as a function of the accuracy as shown in Equation 2.

$$\Pr[|\epsilon| > 1](c, \sigma) = 1 - \mathsf{erf}\left(\frac{3c}{2\sigma\sqrt{2}}\right) = 1 - \mathsf{erf}\left(3\mathsf{erf}^{-1}(\mathsf{accuracy})\right) \tag{2}$$

The probability that a vector $\boldsymbol{e}$ satisfies the ± 1 model equals

$$\begin{aligned}\Pr[\forall\, i \in [\![0; 2t-1]\!]\, |e_i| \leq 1] &= \prod_{i=0}^{2t-1} \Pr[|e_i| \leq 1] = \prod_{i=0}^{2t-1} (1 - \Pr[|\epsilon| > 1](c, \sigma)) \\ &= (1 - \Pr[|\epsilon| > 1](c, \sigma))^{2t} \\ &= \left(\mathsf{erf}(3\mathsf{erf}^{-1}(\mathsf{accuracy}))\right)^{2t}.\end{aligned}$$

To simplify the notations, let ξ be the probability that a vector $\boldsymbol{e}$ fits the ± 1 error model. This allows us to deduce the accuracy as a function of ξ, more precisely, we have:

$$\mathsf{accuracy} = \mathsf{erf}\left(\frac{1}{3}\mathsf{erf}^{-1}(\xi^{\frac{1}{2t}})\right). \tag{3}$$

Since we are given n elements (via their associated lists of Hamming weight) on which this model applies, and considering that our algorithm requires one to recover mt correct elements, we can set the following inequality $n\xi > mt$. Indeed, the average number of noisy Hamming weight lists for the ± 1 error model should be strictly greater than mt. The number of lists, in the ± 1 model, follows a binomial distribution with pmf $\binom{n}{j}(1-\xi)^{n-j}\xi^j$. Hence, the average number of lists in the ± 1 model equals $n\xi$. Table 3 illustrates the different accuracy levels and number of valid lists for the ± 1 model for all the *Classic McEliece* parameter sets.

Table 3: Accuracy and average number of Hamming weight lists in the ± 1 model ($n\xi$) for $\xi \in \{1 - 10^{-2}, 1 - 10^{-3}, 1 - 10^{-4}\}$ for all *Classic McEliece* parameters

ξ	mceliece348864	mceliece460896	mceliece6688128	mceliece6960119	mceliece8192128
			accuracy		
0.9999	0.900372	0.905638	0.909193	0.908305	0.909193
0.9990	0.863789	0.871163	0.876131	0.874892	0.876131
0.9900	0.811925	0.822439	0.829500	0.827741	0.829500
			$n\xi$		
0.9999	3488	4608	6687	6959	8191
0.9990	3485	4603	6681	6953	8184
0.9900	3453	4562	6621	6890	8110

4.3 Error correction algorithm

This part corresponds to line 2 of Algorithm 1 and constitutes the first improvement of the attack presented in [DCV+25]. Once the potentially erroneous lists of Hamming weights $\left(\mathsf{wt}(\alpha^i\beta)\right)_{i\in[\![0;2t-1]\!]}$ have been recovered through a template attack as described in [DCV+25], we use Algorithm 2 to correct those lists.

From here on we will use the following convention. Let $\boldsymbol{x}$ be a list of acquired, potentially erroneous, Hamming weights and $\mathcal{U}_{m,t} = \left\{\left(\mathsf{wt}(\alpha^i\beta)\right)_{i=0}^{2t-1} | (\alpha, \beta \in \mathbb{F}_{2^m}^* \times \mathbb{F}_{2^m}^*)\right\}$ be the set of all lists of Hamming weights contained in the Hamming weight Vandermonde distinguisher. We can thus deduce: $\exists \boldsymbol{y} \in \mathcal{U}_{m,t}, \exists \boldsymbol{r} \in ([\![-1; 1]\!])^{2t}, \boldsymbol{x} = \boldsymbol{y} + \boldsymbol{r}$. The objective of the proposed algorithm is to recover $\boldsymbol{y}$ from $\boldsymbol{x}$. Our solution, as well shall see in the

Algorithm 2 Error correction algorithm

Input: A Hamming weights list $\boldsymbol{x} = (x_i)_{i=0}^{2t-1}$ and the array $\mathcal{U}_{m,t}$
Output: $\boldsymbol{y}_s \in [\![0; m]\!]^{2t}$ or $(\perp, count)$

1: $\mathcal{P}_{sols} = \varnothing$
2: $count = 0$
3: $d_{0min} = 2t + 1$
4: **for** $\boldsymbol{y} \in \mathcal{U}_{m,t}$ **do**
5: **if** $(d_\infty(\boldsymbol{x}, \boldsymbol{y}) = 0)$ **then** ▷ $\boldsymbol{x}$ is correct
6: **return** $\boldsymbol{y}$
7: **else if** $(d_\infty(\boldsymbol{x}, \boldsymbol{y}) = 1)$ **then** ▷ d_∞ filtering
8: $\mathcal{P}_{sols} = \mathcal{P}_{sols} \cup \{\boldsymbol{y}\}$
9: **if** $\#\mathcal{P}_{sols} == 1$ **then** ▷ unique solution
10: **return** $\boldsymbol{y}_s = \mathcal{P}_{sols}[0]$
11: **else**
12: **for** $\boldsymbol{y} \in \mathcal{P}_{sols}$ **do** ▷ selection among all the potential solutions
13: $d' = d_0(\boldsymbol{x}, \boldsymbol{y})$ ▷ d_0 filtering
14: **if** $d' < d_{0min}$ **then**
15: $d_{0min} = d'$
16: $\boldsymbol{y}_s = \boldsymbol{y}$
17: $count = 1$
18: **else if** $d' == d_{0min}$ **then**
19: $count+ = 1$
20: **if** $count == 1$ **then**
21: **return** $\boldsymbol{y}_s$
22: **else** ▷ multiple solutions
23: **return** $(\perp, count)$

next section, is the Maximum likelihood decoder under the ± 1 error model. Let us first describe in details the algorithm.

We first consider, based on the ± 1 error model hypothesis Subsection 4.2, the set of all the potentials solutions, denoted by $\mathcal{P}_{sols}$, which are all the lists of Hamming weights $\boldsymbol{y} = \left(\mathsf{wt}(\alpha^i \beta)\right)_{i=0}^{2t-1}$ with $(\alpha, \beta) \in (\mathbb{F}_{2^m}^*)^2$ such that $\max_{i \in [\![0;2t-1]\!]} |\boldsymbol{y}_i - \boldsymbol{x}_i| = d_\infty(\boldsymbol{y}, \boldsymbol{x}) = 1$. From line 4 to line 8, the set of potential solutions of $\boldsymbol{x}$ is created by computing, for every $\boldsymbol{y} \in \mathcal{U}_{m,t}$, the infinite distance of $\boldsymbol{x}$ and $\boldsymbol{y}$ and including the $\boldsymbol{y}$ with such a distance equal to 1. As long as the hypothesis is respected, we know for sure that the correct Hamming weight list is included in that set. In the case where $\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{r}$ has been acquired without error (i.e. $\boldsymbol{r} = 0$), then the corresponding $\boldsymbol{y} \in \mathcal{U}_{m,t}$ will be found, and returned by the algorithm, during this step as it is the only element of $\mathcal{U}_{m,t}$ with an infinite distance from $\boldsymbol{x}$ equal to 0. Otherwise, we will exploit the $\mathcal{P}_{sols}$ set to recover the correct $\boldsymbol{y}$. Once that set is formed, there are two possibilities :

$\#\mathcal{P}_{sols} = 1$ Only one element $\boldsymbol{y}$ from the $\mathcal{U}_{m,t}$ set fits the condition on the infinite distance. Therefore, the algorithm outputs that element. It is important to notice that, as long as the ± 1 error model hypothesis is respected, then the outputted element is the correct one.

$\#\mathcal{P}_{sols} > 1$ More than one element fit the condition on the infinite distance. Therefore, other methods must be applied to discriminate all the elements of the potential solutions set and find the one which is most likely to be the correct one.

The ideal case is $\#\mathcal{P}_{sols} = 1$. We have run simulations on the toyeliece51220 and mceliece348864 to verify how often we fall under such a case. We repeat a Monte Carlo

simulation, in which we compute the $\mathcal{P}_{sols}$ set for a large number of lists of Hamming weights randomly chosen in $\mathcal{U}_{m,t}$, on which we add a random error vector ($r_i \in \{\pm 1, 0\}$). The conclusion is that the probability of having $\#\mathcal{P}_{sols} = 1$ is lower than 0.063. Therefore, if we want to recover at least t pairs $(\alpha, g(\alpha))$ out of the first mt pairs, we need to add another method to choose the best solution from the $\mathcal{P}_{sols}$ set. If the first step was not enough to recover the initial list of Hamming weight, we assume that, of all the elements from the $\mathcal{P}_{sols}$ set, the most likely to be correct is the one with the less erroneous coordinates (i.e. the one where the error vector $\boldsymbol{r}$ has the lowest Hamming weight). Therefore the Hamming distance is used. From line 12 to line 19, we compute, for each element $\boldsymbol{y}$ of the $\mathcal{P}_{sols}$ set, the Hamming distance $d_0(\boldsymbol{y}, \boldsymbol{x})$ in order to find which element(s) have the minimal distance as well as the number of elements with the minimal distance to $\boldsymbol{y}$. Once this second step has been applied, two outputs are possible:

Zero list This output means that at least two elements from the $\mathcal{P}_{sols}$ set have the same minimal Hamming distance from $\boldsymbol{x}$. In that case, the *count* variable allows to know their numbers. We show experimentally in Subsection 6.2 that this happens with low probability even for highly erroneous lists of Hamming weights.

Unique list In that case, only one element from the $\mathcal{P}_{sols}$ set has the minimal Hamming distance from $\boldsymbol{x}$. We show experimentally in Subsection 6.2 that, even for errors with higher Hamming weight, the outputted list is the correct one with high probability.

4.4 From leakage to communication model

We translate the leakage model into a communication model, following the same framework as in [HRG14]. We define the input alphabet $\mathcal{Y} = \{0, \ldots, m\}$ and output alphabet $\mathcal{X} = \{0, \ldots, m\}$. Hence, the communication channel is $Z = (\mathcal{Y}, \mathcal{X}, \Pr_Z)$ where the stochastic matrix of the channel is described in Equation 4. The parameter a that we consider here is a fairly good approximation of the accuracy.

$$\Pr_Z = \begin{pmatrix} \frac{1+a}{2} & \frac{1-a}{2} & 0 & 0 & \ldots & 0 & 0 & 0 \\ \frac{1-a}{2} & a & \frac{1-a}{2} & 0 & \ldots & 0 & 0 & 0 \\ \vdots & & & & \ddots & & & \vdots \\ 0 & 0 & 0 & 0 & \ldots & \frac{1-a}{2} & a & \frac{1-a}{2} \\ 0 & 0 & 0 & 0 & \ldots & 0 & \frac{1-a}{2} & \frac{1+a}{2} \end{pmatrix} \tag{4}$$

Notice that we have $\Pr_Z(X = 0 \mid Y = 0) = \Pr_Z(X = m \mid Y = m) = a + \frac{1-a}{2}$ which is greater than all the other cases (due to the boundaries of the Gaussian distributions in Figure 1).

Theorem 1. *Algorithm 2 is the Maximum likelihood decoder for the Z channel.*

Proof. Since, Z is memoryless the probability of receiving the $\boldsymbol{x} \in \mathcal{X}^{2t}$ given that $\boldsymbol{y} \in \mathcal{Y}^{2t}$ was sent equals

$$\Pr_Z(\boldsymbol{x} \mid \boldsymbol{y}) = \prod_{i=1}^{2t} \Pr_Z(X = x_i \mid Y = y_i). \tag{5}$$

Now based on the definition of the channel Z we have:

$$\begin{aligned}
\Pr_Z(X = x_i \mid Y = y_i) &= a \cdot \mathbb{1}_{\{x_i=y_i, y_i\in\{1,\ldots,m-1\}\}} + \frac{1+a}{2} \cdot \mathbb{1}_{\{x_i=y_i, y_i\in\{0,m\}\}} \\
&\quad + \frac{1-a}{2} \cdot \mathbb{1}_{\{x_i=y_i-1, x_i\neq 0\}} + \frac{1-a}{2} \cdot \mathbb{1}_{\{x_i=y_i+1, y_i\neq m\}} \\
&\quad + 0 \cdot \mathbb{1}_{\{|x_i-y_i|\geq 2\}} \\
\Pr_Z(\boldsymbol{x} \mid \boldsymbol{y}) &= 0^{\#\{i||y_i-x_i|\geq 2\}} \cdot \left(\frac{1+a}{2}\right)^{\#\{i|x_i=y_i, y_i\in\{0,m\}\}} \cdot a^{\#\{i|x_i=y_i, y_i\neq 0,m\}} \\
&= \left(\frac{1-a}{2}\right)^{\#\{i|x_i=y_i-1, y_i\neq 0\}} \cdot \left(\frac{1-a}{2}\right)^{\#\{i|x_i=y_i+1, y_i\neq m\}} \\
\Pr_Z(\boldsymbol{x} \mid \boldsymbol{y}) &= 0^{\#E_{2,\infty}} \cdot \left(\frac{1+a}{2a}\right)^{\#J_{\boldsymbol{x},\boldsymbol{y}}} \cdot a^{2t-d_0(\boldsymbol{x},\boldsymbol{y})} \cdot \left(\frac{1-a}{2}\right)^{d_0(\boldsymbol{x},\boldsymbol{y})}.
\end{aligned}$$

where $E_{2,\infty} = \{i \mid d_\infty(\boldsymbol{x},\boldsymbol{y}) \geq 2\}$, $J_{\boldsymbol{x},\boldsymbol{y}} = \{i \mid x_i = y_i \in \{0,m\}\}$ and $0^0 = 1$.

Using the fact that $1 + a \geq 2a$ we obtain

$$\Pr_Z(\boldsymbol{x} \mid \boldsymbol{y}) \geq 0^{\#E_{2,\infty}} \cdot a^{2t-d_0(\boldsymbol{x},\boldsymbol{y})} \cdot \left(\tfrac{1-a}{2}\right)^{d_0(\boldsymbol{x},\boldsymbol{y})}, \tag{6}$$

with equality when $\#J_{\boldsymbol{x},\boldsymbol{y}} = 0$ or when $a = 1/3$. The maximum likelihood decoder for the channel Z is an algorithm that receives a vector from $\mathcal{X}^{2t}$ and returns a vector $\boldsymbol{y} \in \mathcal{Y}^{2t}$ s.t.

$$\begin{aligned}
\mathcal{D}_{ML}(\boldsymbol{y}) &= \operatorname{argmax}_{\boldsymbol{y}\in\mathcal{Y}^{2t}} \Pr_Z(\boldsymbol{x} \mid \boldsymbol{y}) \\
&= \operatorname{argmax}_{\boldsymbol{y}\in\mathcal{Y}^{2t}} 0^{\#E_{2,\infty}} \cdot a^{2t-d_0(\boldsymbol{x},\boldsymbol{y})} \cdot \left(\frac{1-a}{2}\right)^{d_0(\boldsymbol{x},\boldsymbol{y})} \cdot \left(\frac{1+a}{2a}\right)^{\#J_{\boldsymbol{x},\boldsymbol{y}}}.
\end{aligned}$$

□

Remark 1. Any pair $(\boldsymbol{x},\boldsymbol{y})$ satisfying $d_\infty(\boldsymbol{x},\boldsymbol{y}) \geq 2$ yields $\Pr_Z(\boldsymbol{y} \mid \boldsymbol{x}) = 0$. For any pair $(\boldsymbol{x},\boldsymbol{y})$ satisfying $d_\infty(\boldsymbol{x},\boldsymbol{y}) < 2$ we have

$$\Pr_Z(\boldsymbol{x} \mid \boldsymbol{y}) \geq a^{2t-d_0(\boldsymbol{x},\boldsymbol{y})} \cdot \left(\tfrac{1-a}{2}\right)^{d_0(\boldsymbol{x},\boldsymbol{y})} = a^{2t} \left(\tfrac{1-a}{2a}\right)^{d_0(\boldsymbol{x},\boldsymbol{y})}. \tag{7}$$

Hence, for any $a \geq 1/3$ the ML decoder searches for $\boldsymbol{y}$ that minimizes $d_0(\boldsymbol{x},\boldsymbol{y})$, also known as the closest vector w.r.t. the d_0 distance.

Remark 2. Given a received vector $\boldsymbol{x} \in \mathcal{X}^{2t}$ an ML decoder should first select the vectors $\boldsymbol{y} \in \mathcal{Y}^{2t}$ which satisfy $d_\infty(\boldsymbol{y},\boldsymbol{x}) = 0$. Then it decodes by choosing the closest vector to $\boldsymbol{x}$. It is thus the closest vector to $\boldsymbol{x}$ w.r.t. the Hamming distance while being inside the closed ball around $\boldsymbol{x}$, $B(\boldsymbol{x},1)_{d_\infty}$.

Beyond the ± 1 error model We have shown that using d_∞ and the Hamming distance one can decode in the ± 1 model. However, one can definitely extend our channel model to ± 2 or even higher error models. The recipe is identical: extend the channel model Z and compute the channel probabilities. By doing so, we can tweak Algorithm 2 by including the distances corresponding to the new channel model.

4.5 Error correction capability

One possible way to characterize the efficiency of the error correction algorithm presented in Subsection 4.3 is to determine its error correction capability denoted as $c_{\min_{m,t}}$ which is defined as the maximum number of coordinates of $\boldsymbol{y}$ that can be erroneous while still being correctable for all $\boldsymbol{y} \in \mathcal{U}_{m,t}$.

Notice that $\boldsymbol{y}$ can be recovered, using Algorithm 2, if and only if $\forall \boldsymbol{y}' \in \mathcal{U}_{m,t} \setminus \{\boldsymbol{y}\}, d_\infty(\boldsymbol{y}', \boldsymbol{x}) \leq 1 \Rightarrow d_0(\boldsymbol{y}, \boldsymbol{x}) < d_0(\boldsymbol{y}', \boldsymbol{x})$. From Property 1, one can deduce that all $\boldsymbol{y}'$ such that $d_\infty(\boldsymbol{y}, \boldsymbol{y}') > 2$ do not affect the recovery of $\boldsymbol{y}$ from $\boldsymbol{x}$ using Algorithm 2. Let $\mathcal{P}_{collisions}(\boldsymbol{y})$ be the set of all $\boldsymbol{y}' \in \mathcal{U}_{m,t} \setminus \{\boldsymbol{y}\}$ such that $d_\infty(\boldsymbol{y}, \boldsymbol{y}') \leq 2$, then one can compute the maximum number of coordinates that can be erroneous while $\boldsymbol{y}$ can still be recovered as $\left\lfloor \frac{\min_{\{\boldsymbol{y}' \in \mathcal{P}_{collisions}(\boldsymbol{y})\}} d_0(\boldsymbol{y}, \boldsymbol{y}') - 1}{2} \right\rfloor$. If the $\mathcal{P}_{collisions}(\boldsymbol{y})$ set is empty for a given element $\boldsymbol{y}$, then it means that $\boldsymbol{y}$ can always be recovered independently of the number of erroneous coordinates.

Property 1. Let $\boldsymbol{x}$ be an erroneous list of Hamming weights such that $\exists \boldsymbol{y} \in \mathcal{U}_{m,t}, \exists \boldsymbol{r} \in (\llbracket -1; 1 \rrbracket)^{2t}, \boldsymbol{x} = \boldsymbol{y} + \boldsymbol{r}$. Then we have $\forall \boldsymbol{y}' \in \mathcal{U}_{m,t} \setminus \{\boldsymbol{y}\}, d_\infty(\boldsymbol{y}, \boldsymbol{y}') > 2 \Rightarrow d_\infty(\boldsymbol{y}', \boldsymbol{x}) > 1$.

Proof. Let $\boldsymbol{x}$ be an erroneous list of Hamming weights such that $\exists \boldsymbol{y} \in \mathcal{U}_{m,t}, \exists \boldsymbol{r} \in (\llbracket -1; 1 \rrbracket)^{2t}, \boldsymbol{x} = \boldsymbol{y} + \boldsymbol{r}$, and $\boldsymbol{y}' \in \mathcal{U}_{m,t} \setminus \{\boldsymbol{y}\}$ such that $d_\infty(\boldsymbol{y}', \boldsymbol{y} + \boldsymbol{r}) \leq 1$ then

$$\begin{aligned} d_\infty(\boldsymbol{y}', \boldsymbol{y}) &\leq d_\infty(\boldsymbol{y}', \boldsymbol{x}) + d_\infty(\boldsymbol{x}, \boldsymbol{y}) \\ &\leq d_\infty(\boldsymbol{y}', \boldsymbol{y} + \boldsymbol{r}) + d_\infty(\boldsymbol{y} + \boldsymbol{r}, \boldsymbol{y}) \\ &\leq d_\infty(\boldsymbol{y}', \boldsymbol{y} + \boldsymbol{r}) + \max_{i \in \llbracket 0; 2t \rrbracket} |\boldsymbol{r}_i| \\ &\leq 2 \end{aligned}$$

So, by contraposition, we have $d_\infty(\boldsymbol{y}', \boldsymbol{y}) > 2 \Rightarrow d_\infty(\boldsymbol{y}', \boldsymbol{x}) > 1$. □

The error correction capability of the algorithm is computed using Algorithm 6 that computes $\left\lfloor \frac{\min_{\{\boldsymbol{y}' \in \mathcal{P}_{collisions}(\boldsymbol{y})\}} d_0(\boldsymbol{y}, \boldsymbol{y}') - 1}{2} \right\rfloor$ for all $\boldsymbol{y} \in \mathcal{U}_{m,t}$ and returns $c_{\min_{m,t}} = \min_{\boldsymbol{y} \in \mathcal{U}_{m,t}} \left\lfloor \frac{\min_{\{\boldsymbol{y}' \in \mathcal{P}_{collisions}(\boldsymbol{y})\}} d_0(\boldsymbol{y}, \boldsymbol{y}') - 1}{2} \right\rfloor$. Table 4 shows the result for the security parameter sets toyeliece51220 and mceliece348864.

Table 4: Error correction capability for the toyeliece51220 and mceliece348864 sets of security parameters.

Parameter set	toyeliece 51220	mceliece 348864
$c_{\min_{m,t}}$	4	5
$\frac{c_{\min_{m,t}}}{2t}$	10%	3,9%

For toyeliece51220, where the lists are composed of $2t = 40$ Hamming weights, the minimal error correction capability $c_{\min_{m,t}}$ is 4. It means that, for every $\boldsymbol{y} \in \mathcal{U}_{m,t}$ we can correct up to 4 erroneous Hamming weights. Concerning mceliece348864 where the lists are composed of 128 Hamming weights, it is possible to correct up to 5 erroneous Hamming weights.

4.6 Noisy Hamming weights lists correction probability

Although it allows to obtain a lower bound in the number of coordinates that can be erroneous while still being able to correct, the error correction capability $c_{\min_{m,t}}$ presented in Subsection 4.5 suffers many drawbacks. In particular, the error correction capability corresponds to the global minimum, i.e., for all $\boldsymbol{y} \in \mathcal{U}_{m,t}$. Indeed, as the code formed by the elements $\boldsymbol{y} \in \mathcal{U}_{m,t}$ is *non-linear*, some $\boldsymbol{y}$ can have more erroneous coordinates than the value of the error correction capability $c_{\min_{m,t}}$ while still being correctable. Moreover, the

value $c_{\min_{m,t}}$ gives us no information on the error positions that makes the erroneous lists of Hamming weights no longer correctable.

To get a more precise characterisation of the efficiency of Algorithm 2, we first define, for $\boldsymbol{y} \in \mathcal{U}_{m,t}$, the correction probability of $\boldsymbol{y}$, called $\Pr_{\text{corr}}(\boldsymbol{y})$, as the probability that, given a random error vector $\boldsymbol{r} \in (\llbracket -1; 1 \rrbracket)^{2t}$, $\boldsymbol{y}$ can be recovered from the erroneous list of Hamming weights $\boldsymbol{x} = \boldsymbol{y} + \boldsymbol{r}$ using Algorithm 2. As the probability on the Hamming weight of the error is not uniform, an error with a low Hamming weight is more likely to happen than an error with a high Hamming weight. That probability is equal to $\Pr(\mathsf{wt}(\boldsymbol{r}) = k) = \binom{2t}{k}(1-a)^k a^{2t-k}$ for $k \in \llbracket 0; 2t \rrbracket$, where $a = \Pr(y_i = x_i | x_i)_{i \in \llbracket 0; 2t-1 \rrbracket}$ is the probability, considered independent of the coordinates, of the classifier to correctly recover the Hamming weight of the ith coordinate, as proved by Lemma 1.

Lemma 1. *For any $k \in \llbracket 0; 2t \rrbracket$ we have:*

$$\Pr(\mathsf{wt}(\boldsymbol{r}) = k) = \binom{2t}{k}(1-a)^k a^{2t-k}. \tag{8}$$

where $a = \Pr(y_i = x_i | x_i)_{i \in \llbracket 0; 2t-1 \rrbracket}$ is the probability, considered independent of the coordinates, of the classifier to correctly recover the Hamming weight.

Proof.

$$\begin{aligned}\Pr(\mathsf{wt}(\boldsymbol{r}) = k) &= \sum_{i+j=k} \binom{2t}{i} \left(\frac{1-a}{2}\right)^i \binom{2t-i}{j} \left(\frac{1-a}{2}\right)^j a^{2t-(i+j)} \\ &= \sum_{i=0}^{k} \binom{2t}{k}\binom{k}{i} \left(\frac{1-a}{2}\right)^k a^{2t-k} = \binom{2t}{k}(1-a)^k a^{2t-k}.\end{aligned}$$

□

Remark 3. Straightforward we notice that on average the Hamming weight is concentrated around the value $(1-a)2t$.

To take that non-uniformity into account in our error correction probability, we consider the conditional probability $\Pr_{\text{corr}}(\boldsymbol{y} | \mathsf{wt}(\boldsymbol{r}))$ as the correction probability of $\boldsymbol{y}$ for a given Hamming weight of the error vector $\boldsymbol{r}$. Using the result from Subsection 4.5, we know that:

$$\forall \boldsymbol{y} \in \mathcal{U}_{m,t}, \forall \boldsymbol{r} \in (\llbracket -1; 1 \rrbracket)^{2t}, \forall i \in \left\llbracket 0; c_{\min_{m,t}} \right\rrbracket, \Pr_{\text{corr}}(\boldsymbol{y} | wt(\boldsymbol{r}) = i) = 1.$$

The value $\Pr_{\text{corr}}(\boldsymbol{y})$ of the correction probability of a given list of Hamming weights $\boldsymbol{y} \in \mathcal{U}_{m,t}$ is given in Equation 9.

$$\begin{aligned}&\Pr_{\text{corr}}(\boldsymbol{y}) = \textstyle\sum_{k=0}^{2t} \Pr(\boldsymbol{y} | wt(\boldsymbol{r}) = k) \Pr(wt(\boldsymbol{r}) = k) \\ &= \sum_{k=0}^{c_{\min_{m,t}}} \binom{2t}{k}(1-a)^k a^{2t-k} + \sum_{k=c_{\min_{m,t}}+1}^{2t} \binom{2t}{k}(1-a)^k a^{2t-k} \Pr(\boldsymbol{y} | wt(\boldsymbol{r}) = k)\end{aligned} \tag{9}$$

In addition, we define, for a given set of parameters of *Classic McEliece*, the correction probability $\Pr_{\text{corr}_{m,t}}$ of Algorithm 2 as the average correction probability of $\boldsymbol{y} \in \mathcal{U}_{m,t}$. The value of $\Pr_{\text{corr}_{m,t}}$ is given in Equation 10.

$$\Pr_{\text{corr}_{m,t}} = \frac{\sum_{\boldsymbol{y} \in \mathcal{U}_{m,t}} \Pr_{\text{corr}}(\boldsymbol{y})}{\#\mathcal{U}_{m,t}} \tag{10}$$

We estimate $\Pr_{\text{corr}_{m,t}}$ experimentally in Subsection 6.2.

5 Exploitation of the recovered Goppa polynomial g

This part corresponds to line 5 of Algorithm 1. In [DCV+25], the authors recovers all the required $mt + \delta$ pairs $(\alpha, g(\alpha))$ at once without exploiting the Goppa polynomial g. In this section, we detail how to exploit the recovered polynomial to decrease both the number of required pairs from $mt + \delta$ random pairs to the first mt pairs and how to ease the recovery of the full permuted support $\mathcal{L}$ using those first mt pairs.

5.1 Recovery of the first mt pairs

Once the Goppa polynomial g has been recovered from t pairs $(\alpha, g(\alpha))$, one no longer needs to look in the complete Hamming weight Vandermonde-like matrix $\beta \boldsymbol{V}_\alpha$ to recover a pair $(\alpha, g(\alpha))$. Indeed, as g is known, there is, for each $\alpha \in \mathbb{F}_{2^m}$, only one possible list of Hamming weights corresponding to $\beta = g(\alpha)$. Therefore, we can extract a new Vandermonde-like matrix $g(\alpha)\boldsymbol{V}_\alpha = \left(\mathsf{wt}(\alpha^i g(\alpha))\right)_{i \in [\![0;2t-1]\!]}$ for $\alpha \in \mathbb{F}_{2^m}^*$ from the initial one. This allow to decrease the time required to recover the remaining pairs $(\alpha, g(\alpha))$ and the number of collisions inside the Vandermonde-like matrix.

For the parameter sets of *Classic McEliece* where $m = 13$, using Proposition 1, we know that, assuming that the lists of Hamming weights of the first mt columns are correct, or have been corrected successfully, we can recover the first mt pairs $(\alpha, g(\alpha))$ from the matrix $g(\alpha)\boldsymbol{V}_\alpha$ as they will not be any list of Hamming weights that belongs to several α's.

For the parameter set mceliece348864 and toyeliece51220, using Algorithm 7 shows that it is still possible, while highly unlikely, to have lists of Hamming weights inside the matrix $g(\alpha)\boldsymbol{V}_\alpha$ that belongs to several α's. In such case, all the potential pairs are saved for the given column and we show in Subsection 5.2 how to exploit the public key $\boldsymbol{T}$ to recover the correct α.

Proposition 1. *For the field proposed in Classic McEliece where $m = 13$, it holds:*

$$\forall (\alpha, \alpha') \in (\mathbb{F}_{2^{13}}^*)^2, \forall (\beta, \beta') \in (\mathbb{F}_{2^{13}}^*)^2,$$
$$\left(\mathsf{wt}(\alpha^i \beta)\right)_{i \in [\![0;2t-1]\!]} = \left(\mathsf{wt}(\alpha'^i \beta')\right)_{i \in [\![0;2t-1]\!]} \Rightarrow \alpha = \alpha'.$$

Proof. We run Algorithm 7 that verifies by exhaustive search that there is no collision between two lists of Hamming weights that belongs to different α's. Using the $\mathcal{U}_{m,t}$ from the mceliece460896 parameter set, with $m = 13$ and $t = 96$, the exhaustive search returns an empty list, which means that there is no list of Hamming weights $\boldsymbol{y} \in \mathcal{U}_{m,t}$ such that there are several pairs $(\alpha, \beta) \in (\mathbb{F}_{2^{13}}^*)^2$ such that $\left(\mathsf{wt}(\alpha^i \beta)\right)_{i \in [\![0;2t-1]\!]} = \boldsymbol{y}$. This result holds for other *Classic McEliece* parameter sets where $m = 13$ as they have a higher value for t. □

5.2 Improved full recovery of the private permuted support $\mathcal{L}$

In [DCV+25], to recover the full permuted support $\mathcal{L}$, they first need to partially compute the matrix $\boldsymbol{H}'$, which is the binary extension of the Vandermonde-like matrix $\boldsymbol{H}$. More precisely, they must find mt columns of $\boldsymbol{H}'$ that are linearly independent. On the assumption that $\boldsymbol{H}'$ is random, the authors shows that, if we are to draw mt columns at random, then the probability for them to be linearly independent is only equal to $\prod_{i=1}^{mt} \left(1 - 2^{-i}\right)$, which gives a probability of approximately 29% for every sets of parameters of *Classic McEliece*. To increase that probability, they propose to recover δ more columns, so that among those $mt + \delta$ columns, mt are linearly independent with a high enough probability.

However, recall that $\boldsymbol{H}_{\text{pub}} = (\boldsymbol{I}_{\text{mt}}|\boldsymbol{T}) = \boldsymbol{E}\boldsymbol{H}'$ with $\boldsymbol{E} \in \mathbb{F}_{2^m}^{mt \times mt}$ the invertible matrix that corresponds to the Gaussian elimination step of the KeyGen algorithm. Let $\mathcal{I}_0 = [\![0; mt-1]\!]$ and $\mathcal{I}_1 = [\![mt; n-1]\!]$, then:

$$\begin{aligned}
&\boldsymbol{E}\boldsymbol{H}' = (\boldsymbol{E}\boldsymbol{H}'[:, \mathcal{I}_0] | \boldsymbol{E}\boldsymbol{H}'[:, \mathcal{I}_1]) = (\boldsymbol{I}_{\text{mt}}|\boldsymbol{T}) \\
\Rightarrow &\boldsymbol{E}\boldsymbol{H}'[:, \mathcal{I}_0] = \boldsymbol{I}_{\text{mt}} \text{ and } \boldsymbol{E}\boldsymbol{H}'[:, \mathcal{I}_1] = \boldsymbol{T} \\
\Rightarrow &\boldsymbol{E}^{-1} = \boldsymbol{H}'[:, \mathcal{I}_0] \text{ and } \boldsymbol{E}^{-1}\boldsymbol{T} = \boldsymbol{H}'[:, \mathcal{I}_0]\boldsymbol{T} = \boldsymbol{H}'[:, \mathcal{I}_1].
\end{aligned}$$

Therefore, with the first mt columns (i.e. the first mt pairs $(\alpha, g(\alpha))$), we can compute the $\boldsymbol{H}'[:, \mathcal{I}_0]$ matrix and multiply it with public key $\boldsymbol{T}$ to get the matrix $\boldsymbol{H}'[:, \mathcal{I}_1]$. We then divide its second row by its first row to find the rest of the permuted support $\mathcal{L}$.

If the lists of Hamming weights of one or more columns of $\boldsymbol{H}'[:, \mathcal{I}_0]$ matches several pairs $(\alpha, g(\alpha))$, we find the permuted support $\mathcal{L}$ by computing $\boldsymbol{H}'[:, \mathcal{I}_0]$ with all possible values for the first mt elements until we find the correct permuted support $\mathcal{L}$.

6 Experimental results

6.1 Template attack

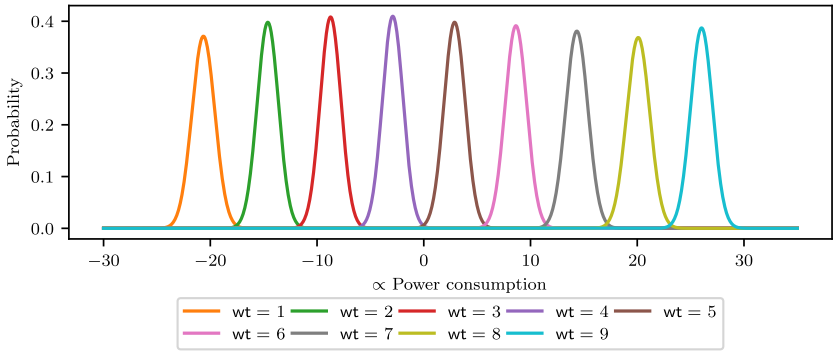
In this section, we show the result of the profiling stage of a template attack to support the ± 1 error model hypothesis made in Subsection 4.2. This attack was performed using a setup identical to [DCV+25]. We use an STM32F303RCT6 microcontroller programmed with the syndrome computation function used in the reference implementation of the decapsulation algorithm that was compiled with `arm-none-eabi-gcc` version `9.2.1` using the `-Os` optimisation. We carry out the attack with both toyeliece51220 and mceliece348864 security parameter sets. For the side-channel acquisitions, we use the open-source ChipWhisperer platform developed by NewAE [OC14]. We built two sets of templates, one for the leakage on the Hamming weights of $g^{-2}(\alpha)$ composed of $1000 \times n$ traces and one for the leakages on the Hamming weights of $\alpha^i g^{-2}(\alpha)$, $i \in [\![1; 2t-1]\!]$ composed of $1000 \times n \times 2t$ traces. We then apply a dimensionality-reduction method, the Linear Discriminant Analysis [SA08], and use the first component for the templates.

Resulting templates for the toyeliece51220 parameter set are shown in Figure 1. As one can see, the Gaussian distributions are overlapping, therefore validating the ± 1 error model hypothesis stated in Subsection 4.2. Similar overlapping of the Gaussian distributions can be observed for the mceliece348864 parameter set.

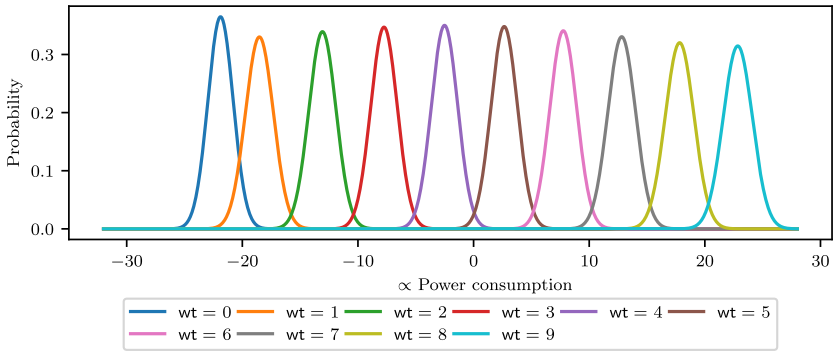
Results of the matching phase are shown in Table 5. As we can see, the probability for the error to be strictly superior to 1 is equal to zero, for both toyeliece51220 and mceliece348864, which confirms our ± 1 error model. Moreover, although there is a difference between $\Pr(y_0 = x_0 | x_0)$ and $\Pr(y_i = x_i | x_i)_{i \in [\![1;2t-1]\!]}$, we consider, to simplify the computation of $\Pr(\mathsf{wt}(\boldsymbol{r}) = k)$, that $\Pr(y_0 = x_0 | x_0) = \Pr(y_i = x_i | x_i)_{i \in [\![1;2t-1]\!]}$.

Table 5: Hamming weight recovery probabilities of the template distinguishers.

Parameter set	toyeliece51220	mceliece348864
$\Pr(y_0 = x_0 \mid x_0)$	0.9943	0.9941
$\Pr(\lvert y_0 - x_0 \rvert = 1 \mid x_0)$	0.0057	0.0059
$\Pr(\lvert y_0 - x_0 \rvert > 1 \mid x_0)$	0	0
$\Pr(y_i = x_i \mid x_i)_{i \in [\![1;2t-1]\!]}$	0.9713	0.9685
$\Pr(\lvert y_i - x_i \rvert = 1 \mid x_i)_{i \in [\![1;2t-1]\!]}$	0.0286	0.0315
$\Pr(\lvert y_i - x_i \rvert > 1 \mid x_i)_{i \in [\![1;2t-1]\!]}$	0	0



(a) Hamming weight Gaussian templates for $\mathsf{wt}(g^{-2}(\alpha))$



(b) Hamming weight Gaussian templates for $\mathsf{wt}(\alpha^i g^{-2}(\alpha))$, $i \in [\![1; 2t-1]\!]$

Figure 1: Visualization of the Hamming weight Gaussian templates for the toyeliece51220 security parameters set, validating the ± 1 error model experimentally.

6.2 Noisy Hamming weights lists correction probability

As already described in Subsection 4.6, given a set of parameters of *Classic McEliece*, the exact computation of the correction probability $\Pr_{\mathrm{corr}_{m,t}}$ requires too much computation. Therefore, we propose in this part an experimental estimation of its value. To do so, we use the Monte Carlo method, with 1 000 000 correction simulations, to approximate the conditional correction probability $\Pr_{\mathrm{corr}_{m,t}}(\mathsf{wt}(\boldsymbol{r}) = k)$, defined in Equation 11, for $k \in [\![0; 2t]\!]$ for the toyeliece51220 and the mceliece348864 sets of parameter.

$$\begin{aligned}
\Pr_{\mathrm{corr}_{m,t}} &= \frac{\sum_{\boldsymbol{y} \in \mathcal{U}_{m,t}} \Pr_{\mathrm{corr}}(\boldsymbol{y})}{\#\mathcal{U}_{m,t}} \\
&= \frac{\sum_{\boldsymbol{y} \in \mathcal{U}_{m,t}} \sum_{k=0}^{2t} \Pr_{\mathrm{corr}}(\boldsymbol{y} | \mathsf{wt}(\boldsymbol{r}) = k) \Pr(\mathsf{wt}(\boldsymbol{r}) = k)}{\#\mathcal{U}_{m,t}} \\
&= \sum_{k=0}^{2t} \frac{\sum_{\boldsymbol{y} \in \mathcal{U}_{m,t}} \Pr_{\mathrm{corr}}(\boldsymbol{y} | \mathsf{wt}(\boldsymbol{r}) = k)}{\#\mathcal{U}_{m,t}} \Pr(\mathsf{wt}(\boldsymbol{r}) = k) \\
&= \sum_{k=0}^{2t} \Pr_{\mathrm{corr}_{m,t}}(\mathsf{wt}(\boldsymbol{r}) = k) \Pr(\mathsf{wt}(\boldsymbol{r}) = k)
\end{aligned} \tag{11}$$

The blue line on Figure 2 shows the approximation of the conditional correction probability $\Pr_{\mathrm{corr}_{m,t}}(\mathsf{wt}(\boldsymbol{r}))$ for toyeliece51220. One can see that the probability is always superior to 99.80% independently of the error's Hamming weight. Moreover, an interesting

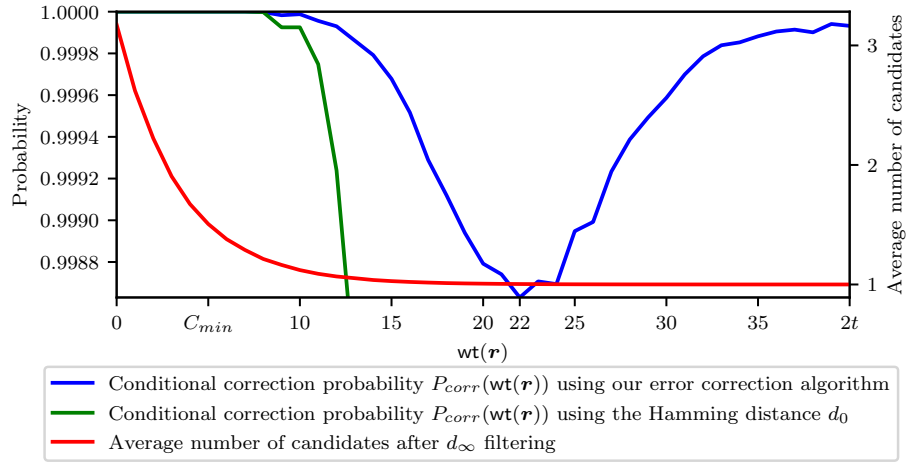


Figure 2: Experimental computation using different distances for error correction for toyeliece51220.

result to notice is that $\Pr_{\mathrm{corr}_{m,t}}(\mathsf{wt}(\boldsymbol{r}))$ decreases with the increase of the Hamming weight of $\boldsymbol{r}$ up to a given value, which is close to t, and then it starts to increase again for higher values of the Hamming weight of $\boldsymbol{r}$. Similar results can be observed on the mceliece348864 set of security parameters where the conditional correction probability starts at 1 for $\mathsf{wt}(\boldsymbol{r}) = 0$, then reaches its minimum of 0.999998 for $\mathsf{wt}(\boldsymbol{r}) = 66 \approx t$ and then increases again. This result can be explained by looking at the behavior of the infinite distance and the Hamming distance separately.

The green line on Figure 2 shows the average decoding probability when only the Hamming distance d_0 is used for correction (i.e. $\mathcal{P}_{collisions}(\boldsymbol{y}) = \mathcal{U}_{m,t} \forall \boldsymbol{y} \in \mathcal{U}_{m,t}$) while the red line corresponds to the average number of potential collisions obtained while approximating $\Pr_{\mathrm{corr}_{m,t}}(\mathsf{wt}(\boldsymbol{r}))$ using the Monte Carlo method. As one can see, when the Hamming weight of the error vector is low, the Hamming distance is enough to correct the erroneous list of Hamming weights while the average number of potential solutions obtained with the d_∞ distance is between 2 and 3, therefore, the d_∞ can not be used alone to correct efficiently. However, when the Hamming weight of the error vector is higher than 8, the Hamming distance d_0 falls drastically and can no longer be used alone to correct efficiently erroneous lists of Hamming weights.

The slight drop in the conditional correction probability of our algorithm is caused by the drastic fall of the Hamming distance starting from $\mathsf{wt}(\boldsymbol{r}) = 8$ while the average number of candidates with distance d_∞ is still superior to 1.

We also approximate the conditional correction probability $\Pr_{\mathrm{corr}_{m,t}}(\mathsf{wt}(\boldsymbol{r}) = k)$, for $k \in [\![0; 2t]\!]$, for the mceliece348864 set of parameter, using the Monte Carlo method with 1 000 000 correction simulations. The obtained conditional correction probability is better than for the toyeliece51220 set of parameter. Indeed, for the mceliece348864 set of parameter, the conditional correction probability is equal to 100% for a Hamming weight of the error lower than 52 and higher than 85. Moreover, in the worst case, which happens when $\mathsf{wt}(\boldsymbol{r}) = 66$, only two lists of Hamming weights could not be corrected, as opposed to 137 for the worst case of toyeliece51220. That better correction is explained by two reasons: there are more coordinates for the lists of Hamming weights of mceliece348864 and the Hamming weights have a wider range of possible values. Those two reasons allow for a better d_∞ filtering, the average number of potential collisions for $\mathsf{wt}(\boldsymbol{r}) = t$ is equal to 1.0068 for toyeliece51220 and 1.000003 for mceliece348864, and so, a better conditional correction probability.

6.3 Algorithm performance

Going from success rate to timing and memory requirements, the performance of our algorithm will be analyzed in this part. Let us first see what happens in the case of noisy Hamming lists that are not included in the ±1 error model.

6.3.1 Detecting larger errors

The proposed method presents a particularly interesting feature, namely, it detects errors that are outside the ±1 model. Simulations show that for Hamming weight list that do not fit in the ±1 error model, the majority of the noisy vectors that have at least one component e_i satisfying $|e_i| \geq 2$ can be detected. Indeed, for such cases we have observed empirically that the majority of the lists $\boldsymbol{x}$ give $\mathcal{P}_{sols} = \emptyset$. In Table 6 we quantify the probability that a list of erroneous Hamming weight lies in the ±1 model, in other words we compute the value ξ, and this for an accuracy of 0.81. Notice that the simulations align with our theoretical estimations from Table 3. In addition, we select from our samples all the vectors that are outside the ±1 model and apply Algorithm 2. If, for a given list $\boldsymbol{x}$, the set $\mathcal{P}_{sols}$ is empty, then we deduce that such errors were detected by our method. The proportion of detectable lists is extremely large, as we can see from Table 6. We do believe that such a capacity of detecting and excluding lists that are outside the ±1 model comes from the value of the accuracy and the block-code (defined by $\beta\boldsymbol{V}$) properties.

Table 6: Proportion of detectable ($\boldsymbol{x}$ with $\mathcal{P}_{sols} = \emptyset$) erroneous lists of Hamming weight outside the ±1 error model for accuracy = 0.81

Parameter set	toyeliece 51220	mceliece 348864	mceliece 460896
ξ (simulation)	0.9966	0.9892	0.9840
Proportion of detectable $\boldsymbol{x}$	0.9251	0.9987	1

6.3.2 Success rate

In this article, we propose a faster algorithmic way to recover the full permuted support $\mathcal{L}$ if we are able to recover the first mt pairs $(\alpha, g(\alpha))$. Here we consider the success rate in the case where the error correction algorithm allows to recover $mt + \delta$ pairs $(\alpha, g(\alpha))$ and then use the algorithmic methods proposed in [DCV+25].

For an accuracy greater than or equal to 0.81, we reach a 100 % success rate, as long as we can detect all Hamming weight lists that are outside the ±1 model. To determine how many columns one needs to sample in order to have at least $mt + \delta$ decodable, we define the following sampling process.

Let ξn be the number of columns that are decodable, and $(1 - \xi)n$ the number of column that are not decodable but detectable by Algorithm 2. Sampling k columns from a total of n, gives the hypergeometric distribution, i.e., $\Pr(X = x) = \frac{\binom{\xi n}{x}\binom{(1-\xi)n}{k-x}}{\binom{n}{k}}$, where X is the random variable counting how many columns are detectable out of k. Hence, we search a value for k such that $\Pr(X \geq mt)$ is reasonably high.

Notice that for all *Classic McEliece* parameters, less than $\delta = 30$ suffices to obtain the required number of decodable elements, as shown in Table 7.

Remark 4. Under the hypothesis that Algorithm 2 detects all non-decodable lists one could decrease the accuracy as low as $accuracy \geq \frac{mt+\delta}{n}$ which is way smaller than the 0.81 accuracy threshold. The only issue is that the accuracy is related to the proportion of non-decodable and non-detectable lists.

Table 7: Success rate ($\Pr(X \geq mt)$) as a function of the number of sampled columns $k = mt + \delta$ for $\xi = 0.99$ or equivalently $\mathsf{accuracy} = 0.81$

	$k = mt + \delta \mid \Pr(X \geq mt)$							
	$\delta = 15$		$\delta = 20$		$\delta = 25$		$\delta = 30$	
mceliece348864	783	0.998147	788	0.999998	793	1.000000	798	1.000000
mceliece460896	1263	0.832299	1268	0.993809	1273	0.999967	1278	1.000000
mceliece6688128	1679	0.361957	1684	0.847677	1689	0.990324	1694	0.999846
mceliece6960119	1562	0.486351	1567	0.910721	1572	0.996008	1577	0.999954
mceliece8192128	1679	0.368679	1684	0.841757	1689	0.988176	1694	0.999741

Although small, when taken into account, the proportion of non-detectable lists has a huge impact on the success rate. Indeed, taking into account for this proportion changes our probability distribution into a multivariate hypergeometric, $\Pr(X_1 = x_1, X_2 = x_2, X_3 = x_3) = \frac{\binom{\xi n}{x_1}\binom{(1-\xi)\nu n}{x_2}\binom{(1-\xi)(1-\nu)n}{x_3}}{\binom{n}{x_1+x_2+x_3}}$, where X_1 is the number of columns that are decodable, X_2 the number of column detectable and non-decodable, and X_3 columns that are non-detectable, non-decodable. Obviously, when $\nu = 1$, in other words all non-decodable are detectable, this brings us back to the hypergeometric distribution.

The success rate becomes now the probability that $X_1 \geq mt$ and $X_3 = 0$ which gives $\Pr(X_1 \geq mt, X_3 = 0) = \sum_{x_2} \sum_{x_1 \geq mt} \frac{\binom{\xi n}{x_1}\binom{(1-\xi)\nu n}{x_2}}{\binom{n}{x_1+x_2}}$. The success rate drops down as the parameter ν decreases. Even in the case of a single non-detectable non-decodable column, there is already an upper bound for the accuracy at 0.772 for the first parameter set, and even lower for the rest of the parameters. Nevertheless, remember that we can use the knowledge of g in order to improve the success of the proposed attack. Hence, we are required to retrieve a considerably smaller quantity of correct Hamming weight lists, more precisely, t.

Table 8: Success rate ($\Pr(X_1 \geq t, X_3 = 0)$) as a function of the number of sampled columns $k = t + \delta$ for $\xi = 0.99$ and $\nu = 0.97$

	$k = t + \delta \mid \Pr(X_1 \geq t, X_3 = 0)$							
	$\delta = 7$		$\delta = 9$		$\delta = 11$		$\delta = 13$	
mceliece348864	71	0.979584	73	0.979070	75	0.978498	77	0.977924
mceliece460896	103	0.999065	105	0.999963	107	0.999999	109	1.000000
mceliece6688128	135	0.974624	137	0.979152	139	0.979199	141	0.978917
mceliece6960119	126	0.996411	128	0.999777	130	0.999990	132	1.000000
mceliece8192128	135	0.978197	137	0.982896	139	0.983013	141	0.982787

6.3.3 Time and memory requirements

Concerning the time complexity, the longest step of Algorithm 2 is the d_∞ filtering (line 4 to line 8). The $\mathcal{P}_{sols}$ set is computed by calculating the infinite distance between the erroneous list $\boldsymbol{x}$ and every $\boldsymbol{y} \in \mathcal{U}_{m,t}$, this requires 2^{2m} comparisons of lists of $2t$ Hamming weights. Assuming $m = \Theta(\log_2 n)$, the time complexity of the algorithm is thus $\mathcal{O}(n^2 \times t)$.

Based on Figure 2, we see that the highest number of potential solutions appears when the Hamming weight of the error is equal to zero. This implies a longer running time for our algorithm. Therefore, we choose to measure the running time for this worst case. The average times for decoding a Hamming weight list, as well as the full permuted support, are given in Table 9. For the toyeliece51220, mceliece348864 and the mceliece460896 set of parameters, the experiments were ran on a Ubuntu laptop with an Intel i5 processor and

64 GB of RAM. However, for the other set of parameters, due to memory limitations, the experiments were ran on a cluster with an Intel Xeon Gold 6444Y and 128 GB of RAM.

Table 9: Average correction time for the lists of Hamming weights for different set of *Classic McEliece* parameters.

Parameter set	toyeliece 51220	mceliece 348864	mceliece 460896	mceliece 6688128	mceliece 6960119	mceliece 8192128
Single list	1.3 ms	74.8 ms	319 ms	521 ms	464 ms	480 ms
Full permuted support $\mathcal{L}$	666 ms	4 min 21 s	24 min 30 s	58 min	54 min	1 h 6 min

We can note that for all different m, our algorithm finishes the correction of a single list of Hamming weights in less than 1 second, making the full attack really efficient in terms of time, with a full permuted support recovery in around 1 hour for the largest set of parameters mceliece8192128.

Concerning the memory requirements, both the list of Hamming weights being corrected and the set of all lists of Hamming weights $\mathcal{U}_{m,t}$ must be loaded in memory. For all *Classic McEliece* parameter, the Hamming weight value is encoded on one byte. The memory usage is therefore approximately equals to $((2^m - 1)^2 + 1) \times 2t$ bytes.

7 Conclusion

In this article, we study the robustness of side-channel attacks on *Classic McEliece* in the presence of side-channel estimation errors. We present a simple yet efficient algorithm that allows us to recover a list of Hamming weights from a list of erroneous Hamming weights, where the error is of at most ± 1 on each Hamming weight, with high probability allowing us to recover the field elements used. We require only t out of n values (or only the mt) or slightly more if we want to cover the recovery failure and interpolate the secret polynomial g, which is the first element of the private key. The error detection can be done by checking the degree of the polynomial. Alternatively, the interpolation with errors method [Ber24] can be used to recover the polynomial from the erroneous evaluation points. The knowledge of g allows us to project the 3-dimensional Hamming weight Vandermonde-like matrix $\beta \boldsymbol{V}_\alpha$ into a 2-dimensional one by fixing $\beta = g(\alpha)$. We use this additional knowledge to recover the mt first columns of the parity check matrix. Since these columns are invertible by construction, this allows us to reduce the number of columns to recover in the side channel attack, and remove the δ margin parameter. The effectiveness of the proposed attack path is demonstrated through various experiments. Our attack shows that the countermeasure proposed in [DCV+25], which consists in shortening the Vandermonde matrix computation, is not a valid approach.

Future works can improve the effectiveness of the search in the Hamming weight Vandermonde-like matrix in terms of efficiency. Our current algorithm has a complexity of $\mathcal{O}(n^2 \times t)$. Perhaps using list decoding for sublists of Hamming weights and performing the intersection of these lists can lead to an efficient unique decoding algorithm. The ± 1 error model we use seems adapted to the side-channel context, where it is more likely to have an error value close to the actual value, but a natural extension is to consider larger errors. The evaluation of the second algorithmic countermeasure of [DCV+25], which proposes to build the extension $\mathbb{F}_{2^m}$ with different polynomials f, could be interesting to evaluate in our scenario. Indeed, they found, for $m = 6$ and $m = 8$, that the polynomial f chosen to build the extension changes the lists of Hamming weights inside the matrix $\beta \boldsymbol{V}_\alpha$ as well as the number of unique lists and therefore the efficiency in recovering each pair $(\alpha, g(\alpha))$ from its lists of Hamming weights $\left(\mathsf{wt}(\alpha^i g(\alpha))\right)_{i \in [\![0;2t-1]\!]}$. This change is likely to affect

the correction properties of the error correction algorithm presented in this article as it will change the distances, both d_∞ and d_0, between the lists of $\beta \boldsymbol{V}_\alpha$. It may also affect the efficiency of the Goppa polynomial g exploitation by making the recovery of the first mt pairs more difficult.

Acknowledgements

This work received funding from the France 2030 program, managed by the French National Research Agency under grant agreement No. ANR-22-PETQ-0008 PQ-TLS and funding from the Aurel Vlaicu University of Arad through the research grant UAV-IRG-1-2025-12.

References

[AAB+22] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. HQC. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[ABB+22] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar-Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, Santosh Ghosh, and Jan Richter-Brokmann. BIKE. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

[ABC+22] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions.

[BCM+25] Marcus Brinkmann, Chitchanok Chuengsatiansup, Alexander May, Julian Nowakowski, and Yuval Yarom. Leaky McEliece: Secret key recovery from highly erroneous side-channel information. *IACR TCHES*, 2025(2):94–125, March 2025. URL: https://tches.iacr.org/index.php/TCHES/article/view/12043, doi:10.46586/tches.v2025.i2.94-125.

[Ber24] Daniel J. Bernstein. Understanding binary-Goppa decoding. *IACR Communications in Cryptology*, 1(1), 2024. doi:10.62056/angy4fe-3.

[Cho14] Omar-Salim Choudary. *Efficient multivariate statistical techniques for extracting secrets from electronic devices*. PhD thesis, University of Cambridge, UK, 2014. URL: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.708342.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Berlin, Heidelberg, August 2003. doi:10.1007/3-540-36400-5_3.

[DCV+25] Vlad-Florin Drăgoi, Brice Colombier, Nicolas Vallet, Pierre-Louis Cayrel, and Vincent Grosso. Full key-recovery cubic-time template attack on classic McEliece decapsulation. *IACR TCHES*, 2025(1):367–391, 2025. URL: https://tches.iacr.org/index.php/TCHES/article/view/11933, doi:10.46586/tches.v2025.i1.367-391.

[GJJ22] Qian Guo, Andreas Johansson, and Thomas Johansson. A key-recovery side-channel attack on classic McEliece implementations. *IACR TCHES*, 2022(4):800–827, 2022. doi:10.46586/tches.v2022.i4.800-827.

[HRG14] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. Good is not good enough - deriving optimal distinguishers from communication theory. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 55–74. Springer, Berlin, Heidelberg, September 2014. doi:10.1007/978-3-662-44709-3_4.

[LDW94] Yuanxing Li, Robert H. Deng, and Xinmei Wang. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *IEEE Trans. Inf. Theory*, 40(1):271–273, 1994. doi:10.1109/18.272496.

[McE78] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. The deep space network progress report 42-44, Jet Propulsion Laboratory, California Institute of Technology, January/February 1978. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

[MMPS09] Amir Moradi, Nima Mousavi, Christof Paar, and Mahmoud Salmasizadeh. A comparative study of mutual information analysis under a Gaussian assumption. In Heung Youl Youm and Moti Yung, editors, *WISA 09*, volume 5932 of *LNCS*, pages 193–205. Springer, Berlin, Heidelberg, August 2009. doi:10.1007/978-3-642-10838-9_15.

[NAB+20] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[Nie86] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986.

[OC14] Colin O'Flynn and Zhizhang (David) Chen. ChipWhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *COSADE 2014*, volume 8622 of *LNCS*, pages 243–260. Springer, Cham, April 2014. doi:10.1007/978-3-319-10175-0_17.

[oST24a] National Institute of Standards and Technology. FIPS203 module-lattice-based key-encapsulation mechanism standard. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf, Aug 2024. Standard.

[oST24b] National Institute of Standards and Technology. FIPS204 module-lattice-based digital signature standard. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.204.pdf, Aug 2024. Standard.

[oST24c] National Institute of Standards and Technology. FIPS205 stateless hash-based digital signature standard. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.205.pdf, Aug 2024. Standard.

[PFH+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[Riv09] Matthieu Rivain. On the exact success rate of side channel analysis in the Gaussian model. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *SAC 2008*, volume 5381 of *LNCS*, pages 165–183. Springer, Berlin, Heidelberg, August 2009. doi:10.1007/978-3-642-04159-4_11.

[SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 411–425. Springer, Berlin, Heidelberg, August 2008. doi:10.1007/978-3-540-85053-3_26.

[SCD+23] Boly Seck, Pierre-Louis Cayrel, Vlad-Florin Dragoi, Idy Diop, Morgan Barbier, Jean Belo Klamti, Vincent Grosso, and Brice Colombier. A side-channel attack against classic McEliece when loading the goppa polynomial. In Nadia El Mrabet, Luca De Feo, and Sylvain Duquesne, editors, *AFRICACRYPT 23*, volume 14064 of *LNCS*, pages 105–125. Springer, Cham, July 2023. doi:10.1007/978-3-031-37679-5_5.

[Sen00] Nicolas Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Trans. Inf. Theory*, 46(4):1193–1203, 2000. doi:10.1109/18.850662.

[SS92] VM Sidelnikov and SO Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Math. AppL*, 2(4):439–444, 1992. doi:10.1515/dma.1992.2.4.439.

A *Classic McEliece* KEM

In 1978, Robert J. McELiece introduced the McEliece cryptosystem as the first code-based cryptosystem [McE78]. The basic idea is to use an error correcting code, relying on a randomly chosen irreducible binary Goppa code, paired with an efficient decoding algorithm. A variant of that cryptosystem, called the Niederreiter cryptosystem [Nie86], was proposed by Harald Niederreiter in 1986 by replacing the generator matrix in the encryption algorithm with a parity-check matrix. Although its original version, based on a Reed-Solomon code, was proven to be insecure [SS92], the use of an irreducible Goppa code is still considered secure. The two cryptosystems have been proven to be equivalent [LDW94].

Hereafter we describe the different algorithms of *Classic McEliece* KEM.

A.1 Algorithms

Like every KEM, *Classic McEliece* is composed of three algorithms: Key generation, Encapsulation, and Decapsulation.

Key generation This algorithm generates both the public and private keys (pk, sk) that will then be used for the encapsulation and decapsulation algorithms. It should be noted that the key-generation algorithm is a highly computationally intensive process; as a

result, the output of the algorithm is considered a long-term secret. Algorithm 3 describes the different steps of the algorithm.

Algorithm 3 The key-generation algorithm of the *Classic McEliece* KEM.

Input: The *Classic McEliece* security parameters (m, n, t)
Output: The private key $\mathsf{sk} = (g, \mathcal{L})$ and the public key $\mathsf{pk} = \boldsymbol{T}$

1: Generate a set $\mathcal{L} = \{\alpha_0, \ldots, \alpha_{n-1}\}$ of random elements of $\mathbb{F}_{2^m}$ with $\#\mathcal{L} = n$
2: Generate an irreducible monic polynomial $g \in \mathbb{F}_{2^m}[x]$ of degree t
3: Compute the $t \times n$ parity-check matrix $\boldsymbol{H}$

$$\boldsymbol{H} = \begin{pmatrix} g^{-1}(\alpha_0) & \ldots & g^{-1}(\alpha_{n-1}) \\ \vdots & \ddots & \vdots \\ \alpha_0^{t-1} g^{-1}(\alpha_0) & \ldots & \alpha_{n-1}^{t-1} g^{-1}(\alpha_{n-1}) \end{pmatrix}$$

4: Transform $\boldsymbol{H}$ to an $mt \times n$ binary matrix $\boldsymbol{H}'$ ▷ Using the $\mathbb{F}_{2^m} \to \mathbb{F}_2^m$ mapping
5: Transform $\boldsymbol{H}'$ in standard-form $\boldsymbol{H}_{\text{pub}} = (\boldsymbol{I}_{mt} \mid \boldsymbol{T})$
6: **return** $\mathsf{sk} = (g, \mathcal{L})$ and $\mathsf{pk} = \boldsymbol{T}$

The mapping on line 4 of Algorithm 3 is done with the representation of $\mathbb{F}_{2^m}$ given by an extension polynomial $f(x)$, provided by the *Classic McEliece* specification [ABC+22] and such that $\deg(f) = m$ and $f(x)$ is irreducible. The transformation in standard-form in line 5 might fail if the first $mt = n - k$ columns of $\boldsymbol{H}'$ are not invertible (i.e. the identity matrix cannot be obtained). In such case, a new permuted support $\mathcal{L}$ is generated until the computation of the identity matrix, and so the standard-form transformation, succeeds.

Encapsulation The encapsulation algorithm takes as input the public key pk and generates both a ciphertext $\boldsymbol{ct}$ and a session key K. The ciphertext corresponds to a syndrome of a random low-weight vector $\boldsymbol{e}$ that is generated during the algorithm. Its steps are described in Algorithm 4.

Algorithm 4 The encapsulation algorithm of the *Classic McEliece* KEM.

Input: The public key $\mathsf{pk} = \boldsymbol{T}$
Output: The ciphertext $\boldsymbol{ct}$ and the session key K

1: Generate a random vector $\boldsymbol{e} \in \mathbb{F}_2^n$ with $\mathsf{wt}(\boldsymbol{e}) = t$
2: Compute $\boldsymbol{ct} = (\boldsymbol{I}_{mt} \mid \boldsymbol{T})\, \boldsymbol{e}^T$
3: Compute $K = \mathsf{hash}(1 | \boldsymbol{e} | \boldsymbol{ct})$
4: **return** $\boldsymbol{ct}$ and K

Decapsulation The decapsulation algorithm decodes the ciphertext $\boldsymbol{ct}$ using the private key sk and to generate the session key K. Its steps are described in Algorithm 5.

Algorithm 5 The decapsulation algorithm of the *Classic McEliece* KEM.

Input: The ciphertext $\boldsymbol{ct}$ and the private key $\mathsf{sk} = (g, \mathcal{L})$
Output: The session key K

1: Compute the vector $\boldsymbol{v} = (\boldsymbol{ct}, 0, \ldots, 0)$ by padding $\boldsymbol{ct}$ with $n - mt$ zeros
2: Compute the parity-check matrix

$$\boldsymbol{H}_{\mathrm{priv}_{g^2}} = \begin{pmatrix} g^{-2}(\alpha_0) & \ldots & g^{-2}(\alpha_{n-1}) \\ \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \ldots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$$

3: Compute the syndrome $\boldsymbol{s} = \boldsymbol{H}_{\mathrm{priv}_{g^2}} \boldsymbol{v}^T$
4: Compute the error locator polynomial $\sigma(x)$ with the Berlekamp-Massey algorithm
5: Compute $(\sigma(\alpha_0), \ldots, \sigma(\alpha_{n-1}))$ for $\alpha_i \in \mathcal{L}$ and recover the error vector $\boldsymbol{e}$
6: Compute $K = \mathsf{hash}(1|\boldsymbol{e}|\boldsymbol{ct})$
7: **return** K

B Error correction capability computation

Algorithm 6 Error correction capability computation algorithm

Input: The array $\mathcal{U}_{m,t}$
Output: $c_{\min_{m,t}}$

1: $c_{\min_{m,t}} = 2t$
2: **for** $\boldsymbol{y} \in \mathcal{U}_{m,t}$ **do**
3: $\quad \mathcal{P}_{collisions}(\boldsymbol{y}) = \{\boldsymbol{y}' \in \mathcal{U}_{m,t} \setminus \{\boldsymbol{y}\} | d_\infty(\boldsymbol{y}', \boldsymbol{y}) \leq 2\}$ ▷ d_∞ filtering
4: $\quad$ **if** $\#\mathcal{P}_{collisions}(\boldsymbol{y}) > 0$ **then**
5: $\quad\quad c = \left\lfloor \frac{\min_{\boldsymbol{y}' \in \mathcal{P}_{collisions}(\boldsymbol{y})} d_0(\boldsymbol{y}, \boldsymbol{y}') - 1}{2} \right\rfloor$ ▷ d_0 filtering
6: $\quad\quad$ **if** $c < c_{\min_{m,t}}$ **then**
7: $\quad\quad\quad c_{\min_{m,t}} = c$
8: **return** $c_{\min_{m,t}}$

C Collisions computation in the reduced Hamming weight Vandermonde-like matrix

Algorithm 7 Collisions between α's.

Input: The array $\mathcal{U}_{m,t}$
Output: $\mathcal{C}ol_\alpha$: the list of all collisions between α's

$\mathcal{C}ol_\alpha$=[]
for $\boldsymbol{y} \in \mathcal{U}_{m,t}$ **do**
    $\mathcal{L}ist_\alpha = []$
    for $\alpha \in (\mathbb{F}_{2^m}^*)$ **do**
        if $\exists \beta \in \mathbb{F}_{2^m}, \left(\mathsf{wt}(\alpha^i \beta)\right)_{i \in [\![0;2t-1]\!]} = \boldsymbol{y}$ **then**
            $\mathcal{L}ist_\alpha$ += $[\alpha]$
    if len($\mathcal{L}ist_\alpha$) > 1 **then**
        $\mathcal{C}ol_\alpha$ += $[\mathcal{L}ist_\alpha]$
return $\mathcal{C}ol_\alpha$