IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–25. DOI:XXXXXXXX

Full Key-Recovery Cubic-Time Template Attack on *Classic McEliece* Decapsulation

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

[1] Faculty of Exact Sciences, Aurel Vlaicu University, Arad, Romania, vlad.dragoi@uav.ro
[2] LITIS, University of Rouen Normandie, Saint-Etienne du Rouvray, France
[3] Université Jean Monnet Saint-Etienne, CNRS, Institut d Optique Graduate School, Laboratoire Hubert Curien UMR 5516, F-42023, SAINT-ETIENNE, France,
{b.colombier;nicolas.vallet;pierre.louis.cayrel;vincent.grosso}@univ-st-etienne.fr

Abstract. *Classic McEliece* is one of the three code-based candidates in the fourth round of the NIST post-quantum cryptography standardization process in the Key Encapsulation Mechanism category. As such, its decapsulation algorithm is used to recover the session key associated with a ciphertext using the private key. In this article, we propose a new side-channel attack on the syndrome computation in the decapsulation algorithm that recovers the private key, which consists of the private Goppa polynomial g and the permuted support $\mathcal{L}$. The attack relies on both practical aspects and theoretical contributions, namely that the side-channel distinguisher can accurately discriminate elements of the permuted support $\mathcal{L}$, while relying only on a standard noisy Hamming weight leakage assumption and that there exists a cubic-time algorithm that uses this information to recover the private Goppa polynomial g. Compared with previous work targeting the *Classic McEliece* private key, this drastically improves both on the assumptions made in the attacker model and on the overall efficiency of the key-recovery algorithm. We have carried out the attack in practice on a microcontroller target running the reference implementation of *Classic McEliece*, and make the full attack source code available.

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

1 Introduction

The potential emergence of the quantum computer in the years to come poses an undeniable threat to the current asymmetric cryptosystems, such as RSA or ECDSA, which are based on number theory problems. Indeed, in 1994, Peter Shor proposed polynomial-time quantum algorithms to solve these problems [Sho94]. Shor's algorithm can factor large integers and compute discrete logarithms efficiently, breaking the security assumptions underlying RSA and ECDSA. While the threat remained distant for many years, recent technological advancements and renewed interest in the field have made *store-now-decrypt-later* attacks increasingly serious. In such attacks, an adversary can store encrypted data now, expecting to decrypt it later when quantum computers become available.

In response to this emerging threat, the U.S. National Institute of Standards and Technology (NIST) initiated a standardization process called the Post-Quantum Cryptography (PQC) Standardization in 2016. This initiative aims at identifying and standardizing so-called post-quantum cryptographic algorithms, that are resistant to quantum attacks. After three rigorous evaluation rounds, CRYSTALS-Kyber [SAB+20] emerged as the sole candidate selected in the public key encryption and Key Encapsulation Mechanism (KEM) category. CRYSTALS-Kyber is based on the hardness of the learning-with-errors problem, a well-studied problem in lattice-based cryptography.

However, three other candidates remain for a fourth round: BIKE [ABB$^+$22], *Classic McEliece* [ABC$^+$22], and HQC [AAB$^+$22]. They are all based on hard problems from coding theory. BIKE and HQC rely on the hardness of decoding quasi-cyclic codes while *Classic McEliece* is based on the hardness of decoding random linear codes.

In this article, we focus on the *Classic McEliece* KEM. It is a code-based cryptosystem which, despite its name, is actually based on the Niederreiter cryptosystem [Nie86]. The security of *Classic McEliece* is rooted in the difficulty of decoding random linear codes, a problem that has withstood decades of cryptanalytic efforts.

Historically, code-based cryptosystems were subject to side-channel attacks even before NIST initiated the PQC standardization process [HMP10, MSSS11, AHPT11, CEvMS16]. Side-channel attacks exploit physical information that can be measured during the execution of cryptographic algorithms on physical systems, such as timing, power consumption, or electromagnetic emissions. Researchers have developed various countermeasures to protect code-based cryptosystems from these attacks, including masking techniques, constant-time implementations, and error detection mechanisms.

As a side benefit of the NIST PQC standardization process, several embedded software and hardware implementations of *Classic McEliece* have been published [CC21, CCD$^+$22, FLZG24]. These implementations are designed to provide practical security against both classical and quantum adversaries. They are optimized for performance on a variety of platforms, from low-power embedded devices to high-performance servers. The continued development and analysis of these implementations are crucial for ensuring the robustness and efficiency of *Classic McEliece* in real-world applications.

High-level description of *Classic Mceliece* Like any KEM, *Classic McEliece* is built on three algorithms: Key generation, Encapsulation and Decapsulation.

To generate a private/public key pair $(\mathsf{sk}, \mathsf{pk})$, one generates a Goppa polynomial $g(x)$ and selects a subset of permuted elements of $\mathbb{F}_{2^m}$, known as the Goppa support $\mathcal{L}$. The private key is then $(g(x), \mathcal{L})$, which defines the private Goppa code. Given a parity-check matrix of this code $\boldsymbol{H}'$ one can deduce the public key $\boldsymbol{H}_{\text{pub}}$, by computing the standard form of $\boldsymbol{H}'$ by Gaussian Elimination. Getting sk from pk boils down to finding both g and $\mathcal{L}$.

To encapsulate a key, one encrypts a message à la Niederreiter, i.e. computes $\boldsymbol{z} = \boldsymbol{H}_{\text{pub}}\boldsymbol{e}^T$ where $\boldsymbol{e}$ is a binary vector of weight $\mathsf{wt}(\boldsymbol{e}) = t$. The resulting key is the hash value of $\boldsymbol{e}$ and $\boldsymbol{z}$. The decapsulation process decodes $\boldsymbol{z}$ using $g(x)$ and $\mathcal{L}$. This step implies the computation of a *syndrome*, solving a key equation and finding the roots of the error locator polynomial $\sigma(x)$. Optimized implementations use the Fast Fourier Transform (FFT) for syndrome computation and roots extraction [CC21].

Related Work The smaller parameters of *Classic McEliece*, compared with other code-based cryptosystems, coupled with the advent of larger embedded devices, have encouraged the advancement of both implementations and attacks on implementations of *Classic McEliece*.

A few key-recovery side-channel attacks on *Classic McEliece* have already been published. Guo et al. presented a key-recovery side-channel attack during the decapsulation step [GJJ22]. Their attack requires to observe the decapsulation of n invalid ciphertexts, where n is the security parameter of *Classic McEliece*. Seck et al. presented a partial key-recovery side-channel attack during the decapsulation. In their work, only the private polynomial g is recovered, from side channel observation while the polynomial coefficients are loaded [SCD$^+$23]. Brinkmann et al. posted a preprint presenting a key-recovery

Table 1: Comparison of key-recovery attacks on the *Classic McEliece* KEM.

Article	Step (Algorithm)	Scenario	Attack type	Implement. and hardware
[BCM+23]	Gauss. Elim. (KeyGen)	1 trace [a]	Side-channel (sim.) Full key recovery	Ref. & Botan N/A
[GJJ22]	FFT $\sigma(x)$ (Decaps.)	n traces $\mathsf{wt}(\boldsymbol{e}) = 1$	Side-channel Full key recovery	Ref. & optim. FPGA & ARM
[SCD+23]	$\mathsf{load}(g)$ (Decaps.)	1 trace [b]	Side-channel Partial key recov.	Optimized ARM
[PGD+22]	$\sigma(x)$ / Valid. (Decaps.)	no CCA2 Faults in σ	Fault inject. (sim.) Alternate key recov.	Reference RISC-V RTL
This article	Syndrome comp. (Decaps.)	1 trace [c]	Side-channel Full key recovery	Reference ARM

[a] Assuming a maximum 0.4 bit-flip error probability

[b] On the ChipWhisperer [OC14]

[c] On the ChipWhisperer [OC14] and with a Hamming weight classifier accuracy higher than 0.945

side-channel attack during the key generation [BCM+23]. They rely on software-simulated leakages that could occur during the Gaussian elimination.

It is noteworthy that these practical works have led authors to propose novel algorithms for attacking code-based cryptography. Kirshanova and May presented several methods for performing a full key recovery against *Classic McEliece*, assuming that an attacker has access to various parts of the secret [KM23]. These parts are referred to as “hints”. However, the actual means by which an attacker could obtain these hints is not discussed, particularly with respect to their practical feasibility in the context of physical attacks.

Contributions

First side-channel full key-recovery attack against the syndrome computation in the reference *Classic McEliece* implementation with a realistic attacker model In the first step of the decapsulation, one computes the syndrome vector given the received ciphertext and the private parity-check matrix. In [CC21] an optimized implementation of this step, based on the Fast Fourier Transform was proposed. However, as pointed out in [GJJ22] such implementations leaks information about the private key. Hence, we focus on the syndrome computation as performed in the reference implementation of the *Classic McEliece* KEM. We carry our attack in practice on all *Classic McEliece* parameter sets and show that, when implemented on the arguably low-noise ChipWhisperer [OC14] platform, a *single* side-channel trace allows one to completely retrieve the private key of the scheme. Compared with existing key-recovery attacks on the *Classic McEliece* we stress out two main advantages of our approach.

1. Compared with [SCD+23], which has an exponential time complexity, the proposed attack relies on a cubic-time algorithm. On a laptop, the private key is recovered in a few seconds for the lowest security parameters and 5 min for the highest.

2. Compared with [BCM+23], the attacker model we consider is much more realistic. We target the decapsulation step which is performed every time a session key in needed, whereas [BCM+23] targets the key generation algorithm, a more complex scenario to be set up in practice.

Hamming weight Vandermonde distinguisher We show how to use a general Hamming weight leakage model applied to the syndrome computation in order to retrieve pairs of elements of the form $(\alpha, g(\alpha))$ where α are elements in the support $\mathcal{L}$ of the private Goppa code. For that, we define a new distinguisher for the elements of the finite field $\mathbb{F}_{2^m}$. More precisely, we show that, with high probability, one can identify each pair $(\alpha, g(\alpha))$ by means of its Vandermonde-type weight vector $\left(g^{-2}(\alpha), \alpha g^{-2}(\alpha), \dots, \alpha^d g^{-2}(\alpha)\right)$ for a sufficiently large integer d.

A cubic-time optimized algorithm for breaking Goppa codes with hints In the literature, two distinct scenarios for breaking Goppa codes with hints exist: knowing the Goppa polynomial $g(x)$ or knowing a set of elements α in the private Goppa support $\mathcal{L}$. The first scenario is exploited in [BCM$^+$23] by means of the well-known Support Splitting Algorithm [Sen00], while the second was proposed in [KM23]. Here, not only do we recover a set of points $(\alpha, g(\alpha))$ by means of the proposed side-channel distinguisher, we also describe a new cubic-time algorithm for breaking Goppa codes with hints, which is more efficient than state-of-the-art solutions. Its time complexity is $\mathcal{O}((t \log_2 n)^3)$, which becomes $\mathcal{O}(n^3)$ under the assumption that $t = n/\log_2 n$.

We summarize the different steps of the attack in Algorithm 1, and give in comments the associated sections in the paper. The complexity of the main steps in analyzed in Subsection 3.5.

Algorithm 1 Overview of the proposed attack.

```
Input: A 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 $\left(\alpha^i g^{-2}(\alpha)\right)_{i=0}^{2t-1} \forall \alpha \in \mathcal{L}$    ▷ Subsection 3.2
2: Recover $mt + \delta$ pairs $(\alpha, g(\alpha))$    ▷ Subsubsection 3.3.1
3: Recover polynomial $g$ from $t$ pairs $(\alpha, g(\alpha))$ via interpolation    ▷ Subsubsection 3.3.2
4: Construct the Vandermonde matrix $\boldsymbol{V}$ using $g$ and the $mt + \delta$ pairs    ▷ Subsection 3.4
5: Compute the change-of-basis $\boldsymbol{S}$ using $\boldsymbol{V}$ and $\boldsymbol{H}_{\text{pub}}$
6: Recover $\boldsymbol{H}_{\text{priv}_g} = \boldsymbol{S}^{-1} \boldsymbol{H}_{\text{pub}}$
7: Recover the full permuted support $\mathcal{L} = \frac{\boldsymbol{H}_{\text{priv}_g}[1:]}{\boldsymbol{H}_{\text{priv}_g}[0:]} \left(= \frac{\alpha_j g(\alpha_j)}{g(\alpha_j)}\right)$
```

Remark 1. From line 2 in Algorithm 1, it is possible to use $mt + 1$ elements α and the method presented by Kirshanova and May [KM23]. However, since the proposed side-channel attack recovers both α and $g(\alpha)$ for a large number of α, we propose a more efficient technique adapted to this additional information.

Organization This article is organized as follows: Section 2 presents the necessary background, providing definitions and notations from coding theory and code-based cryptography. Section 3 describes the different steps of the attack. Section 4 provides experimental results. Countermeasures are discussed in Section 5 and Section 6 concludes the article.

Reproducibility We make the proposed attack fully reproducible by providing the firmware source code and compilation scripts used to program the target microcontroller, as well as scripts that record the side-channel trace and carry out the attack. In case someone wants to reproduce the attack but does not own a ChipWhisperer [OC14], we also provide a way to simulate the side-channel trace following a Hamming weight leakage with an accuracy that can be adjusted. This is available on a GitLab repository.[1]

[1] https://gitlab.univ-st-etienne.fr/sesam/tches-2025-1-full-key-recovery-cubic-time-template-attack-on-classic-mceliece-decapsulation

2 Background

2.1 Notations

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

Integers are denoted using lowercase letters, while intervals of integers will be denoted by $[\![a;b]\!]$. We employ bold uppercase letters for matrices and bold lowercase letters for vectors. Let $\boldsymbol{v}$ be a vector, then v_i represents its i^{th} coordinate and $\boldsymbol{v}^T$ its transpose. Also, we shall denote by $\sup(\boldsymbol{v})$ the set of positions where $v_i \neq 0$.

Let $\boldsymbol{H}$ be a matrix, then h_{ij} is the coefficient of $\boldsymbol{H}$ located at the i^{th} row and j^{th} column. A sub-matrix of $\boldsymbol{H}$ indexed by a set of rows $\mathcal{I}$ and a set of columns $\mathcal{J}$ will be denoted by $\boldsymbol{H}[\mathcal{I},\mathcal{J}]$. Also, a row or a column of a matrix $\boldsymbol{H}$ will be denoted by $\boldsymbol{H}[i:]$ and $\boldsymbol{H}[:j]$, respectively.

Let $\mathcal{A}$ be a set, then $\#\mathcal{A}$ is the number of elements of $\mathcal{A}$. Finite fields of order q will be denoted by $\mathbb{F}_q$. 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 $\deg(f) = m$. Hence, given α a primitive root of $f(x)$, i.e. $\mathbb{F}_{2^m} = \langle\alpha\rangle$ (α is generator), we naturally associate to any element $\beta \in \mathbb{F}_{2^m}$ its vector form in $\mathbb{F}_2$ by $\boldsymbol{\beta} = (\beta_0,\ldots,\beta_{m-1}) \in \mathbb{F}_2^m$ with $\beta = \sum_{i=0}^{m-1} \beta_i\alpha^i$.

2.2 Code-based cryptography

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

A constructive way of defining a linear code $\mathscr{C}$ is by setting a basis, which we call a *generator matrix* ($\boldsymbol{G}$) for $\mathscr{C}$, where $\mathsf{Rank}(\boldsymbol{G}) = k$. We say that a generator matrix $\boldsymbol{G}$ is in standard form if $\boldsymbol{G} = (\boldsymbol{I}_{\mathrm{k}} \mid \boldsymbol{V})$. While having many generator matrices, a code can possess a single standard-form generator matrix.

Definition 2 (**Hamming weight**)**.** Let $\mathscr{C}$ be a $[n,k]_q$ linear code and $\boldsymbol{c} \in \mathscr{C}$ be a codeword. Then the Hamming weight of $\boldsymbol{c}$ is defined by $\mathsf{wt}(\boldsymbol{c}) = \#\{i \in [\![0;n-1]\!]\,|c_i \neq 0\}$.

Definition 3 (**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 called a parity-check matrix of $\mathscr{C}$.
Also, for any $\boldsymbol{v} \in \mathbb{F}_q^n$, one can define the syndrome of the vector $\boldsymbol{v}$ with respect to $\boldsymbol{H}$ by the vector $\boldsymbol{s} = \boldsymbol{H}\boldsymbol{v}^T$.

Notice that a code admits several parity-check matrices, since there are as many bases for the code as there are invertible matrices of size k over $\mathbb{F}_q$. However, as we shall quickly see particular bases/parity-check matrices lead to efficient decoding algorithms. Before delving into decoding strategies, let us introduce the core code family studied in this article, the binary irreducible Goppa codes [Gop70].

Definition 4 (**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 called a Goppa code with parameters $\mathcal{L}$ and $g(x)$.

In the literature, the parameter $g(x)$ is called the Goppa polynomial, and when $g(x)$ is irreducible, the code is known as a binary irreducible Goppa code. The code parameters are n for its length and $k = n - mt$ for its dimension. Hence, a parity-check matrix for the binary Goppa code has mt rows.

The degree of the Goppa polynomial has a significant relevance for the code properties, more precisely, it gives the decoding capacity of the Goppa code. Because we are dealing with a cryptographic context, we consider unique decoding strategies. In the case of Goppa codes, there are at least three well-known decoding algorithms: Berlekamp-Massey [Ber15, Mas69], Patterson [Pat75], or the latest interpolation with errors for Goppa decoding by D. J. Bernstein [Ber24].

The unique decoding problem refers to recovering two vectors $\boldsymbol{e}, \boldsymbol{c} \in \mathbb{F}_2^n$ from $\boldsymbol{z} = \boldsymbol{c} + \boldsymbol{e}$ assuming $\mathsf{wt}(\boldsymbol{e}) \leq t$ and $\boldsymbol{c} \in \mathscr{G}(g, \mathcal{L})$. All decoders mentioned previously start by computing the syndrome polynomial $s_{\boldsymbol{z}}(x) = \sum_{i=0}^{n-1} \frac{z_i}{x-\alpha_i}$ and then solve a key equation in order to retrieve the *error locator polynomial* $\sigma_{\boldsymbol{e}}(x) = \prod_{i \in \mathsf{sup}(\boldsymbol{e})}(x - \alpha_i)$. While the Patterson algorithm primarily employs $g(x)$ for the modular operations, Berlekamp-Massey and Bernstein methods use $g^2(x)$ in order to achieve the same decoding capacity. Consequently, the parity-check matrix for Patterson decoding has t rows while the parity-check matrix of the other decoders has $2t - 1$ rows. More details can be found in [Ber24].

Now, let us move to the core application of the Goppa codes, namely code-based cryptography, with a particular emphasis on the *Classic McEliece* KEM. It should be noted that in several implementations of *Classic McEliece*, in particular in the reference implementation, the Berlekamp-Massey decoder is used.

2.3 The *Classic McEliece* KEM

Introduced in 1978 by Robert J. McEliece, the McEliece cryptosystem was the first code-based cryptosystem [McE78]. It relies on a randomly chosen binary irreducible Goppa code. Its security is based on the hardness of decoding a general linear code, which is $\mathcal{NP}-$complete [BMvT78]. In 1986, Harald Niederreiter proposed a variant of the McEliece cryptosystem, called the Niederreiter cryptosystem [Nie86], that uses a parity-check matrix for the encryption algorithm instead of a generator matrix. In its original version, the Niederreiter cryptosystem used generalized Reed-Solomon codes, but it was proved to be insecure in [SS92]. However, when using binary Goppa instead of Reed-Solomon codes, it is equivalent, in terms of security, to the McEliece cryptosystem [LDW94].

Classic McEliece [ABC+22] is a KEM based on the Niederreiter cryptosystem. The different sets of parameters for *Classic McEliece* are shown in Table 2. This table also includes a smaller parameter set, referred to as toyeliece which has already been considered in prior works [BCM+23].

Table 2: *Classic McEliece* [ABC+22] and `toyeliece` [BCM+23] parameter sets.

Parameter set	toyeliece 51220	mceliece 348864	mceliece 460896	mceliece 6688128	mceliece 6960119	mceliece 8192128
m	9	12	13	13	13	13
n	512	3488	4608	6688	6960	8192
t	20	64	96	128	119	128

2.3.1 Algorithms

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

Key generation This algorithm generates the pair $(\mathsf{pk}, \mathsf{sk})$ of public and private keys used by the encapsulation and decapsulation algorithms. Algorithm 2 describes the different steps of the algorithm.

Algorithm 2 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 $\mathcal{L} = \{\alpha_0, \dots, \alpha_{n-1}\}$ 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 $\boldsymbol{H}$ [2]
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 $\mathsf{pk} = \boldsymbol{T}$
```

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

Line 4 in Algorithm 2 is completed using the representation of $\mathbb{F}_{2^m}$ given by a fixed extension polynomial $f(x)$, where $\deg(f) = m$ and $f(x)$ is irreducible. Regarding line 5, there might be cases where on the first $mt = n - k$ positions $\boldsymbol{H}'$ is not invertible, and thus one can not compute the identity block. In this case, permute some columns and implicitly α_i until we obtain an invertible block (see [ABC$^+$22]). The reordered elements α_i are part of the private key.

Encapsulation The encapsulation algorithm, shown in Algorithm 3, generates both a ciphertext $\boldsymbol{z}$ and a session key K.

Algorithm 3 The encapsulation algorithm of the *Classic McEliece* KEM.

```
Input: The public key $\mathsf{pk} = \boldsymbol{T}$
Output: The ciphertext $\boldsymbol{z}$ 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 $\boldsymbol{z} = (\boldsymbol{I}_{mt} \mid \boldsymbol{T})\, \boldsymbol{e}^T$
3: Compute $K = \mathsf{hash}(1|\boldsymbol{e}|\boldsymbol{z})$
4: return $\boldsymbol{z}$ and $K$
```

Typically, lines 1 and 2 of Algorithm 3 are the Niederreiter encryption algorithm [Nie86]. The ciphertext $\boldsymbol{z}$ allows the holder of the private key to compute a session key that is identical to the one returned by the encapsulation algorithm.

Decapsulation The decapsulation algorithm, shown in Algorithm 4, computes a syndrome polynomial using the Vandermonde basis of the Goppa code to decode the ciphertext $\boldsymbol{z}$ and derive the session key K.

[2] We have simplified the notations and used $\boldsymbol{H}$ instead of $\boldsymbol{H}_{g,\mathcal{L}}$ although the latter is a more appropriate way of identifying the parity-check matrix since $\boldsymbol{H}$ depends on both g and $\mathcal{L}$.

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

```
Input: The ciphertext $\boldsymbol{z}$ and the private key $\mathsf{sk} = (g, \mathcal{L})$
Output: The session key $K$
Compute the vector $\boldsymbol{v} = (\boldsymbol{z}, 0, \ldots, 0)$ by padding $\boldsymbol{z}$ with $n - mt$ zeros
Compute the parity-check matrix
    $\boldsymbol{H}_{\mathrm{priv}_{g^2}} = \begin{pmatrix} g^{-2}(\alpha_0) & \ldots & g^{-2}(\alpha_{n-1}) \\ \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \ldots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$
Compute the syndrome $\boldsymbol{s} = \boldsymbol{H}_{\mathrm{priv}_{g^2}} \boldsymbol{v}^T$
Compute the error locator polynomial $\sigma(x)$ with the Berlekamp-Massey algorithm
Compute $(\sigma(\alpha_0), \ldots, \sigma(\alpha_{n-1}))$ for $\alpha_i \in \mathcal{L}$ and recover the error vector $\boldsymbol{e}$
Compute $K = \mathsf{hash}(1|\boldsymbol{e}|\boldsymbol{z})$
return $K$
```

2.3.2 Security aspects

Let us briefly recall the security arguments on which the public key - private key relation is built on. The *private* Goppa code is the code specified by $g(x)$ and the set of points $\mathcal{L} \subseteq \mathbb{F}_{2^m}$. The *public* Goppa code is defined by the parity-check matrix $\boldsymbol{H}_{\mathrm{pub}}$. As presented in Algorithm 2 one has $\boldsymbol{S}\boldsymbol{H}' = \boldsymbol{H}_{\mathrm{pub}}$ where $\boldsymbol{S}$ is a mt-square binary invertible matrix. This is computed by means of Gaussian elimination, as shown in line 5 in Algorithm 2. Moreover, the elements in $\mathcal{L}$ are chosen from $\mathbb{F}_{2^m}$ in a random order. Therefore, although anyone can compute the set of elements $\mathbb{F}_{2^m}$ (e.g. $\{\alpha^i \mid i \in [\![0; 2^m - 2]\!]\}$ where α is a primitive element of $\mathbb{F}_{2^m}$) the order of these elements in $\mathcal{L}$ is unknown. We thus face a classical code equivalence problem, in this particular case, the Goppa code equivalence problem [Sen00].

		Conditions[3]
Given:	$\boldsymbol{H}_{\mathrm{pub}}, \mathbb{F}_{2^m} = \{\alpha^i \mid i \in [\![0; 2^m - 2]\!]\}$	$\mathcal{L}' \subseteq \mathbb{F}_{2^m}, \mathcal{L}^\pi = \mathcal{L}'$
Output:	$g(x) \in \mathbb{F}_{2^m}[x], \mathcal{L}$	$\boldsymbol{H}_{\mathrm{pub}} = \boldsymbol{S}\boldsymbol{H}^\pi_{g,\mathcal{L}'}$

Notice that, since $g(x)$ is unknown, one is required to enumerate all possible irreducible polynomials of degree t over $\mathbb{F}_{2^m}$ in order to construct the matrix $\boldsymbol{H}_{g,\mathcal{L}'}$. There are $\frac{1}{t} \sum_{d|t} \mu\left(\frac{t}{d}\right) 2^{md} \sim \frac{2^{mt}}{t}$ such polynomials (μ denotes the Möbius function) [Lot02]. This is infeasible in practice for all proposed parameter sets.

One can look at the problem from a general point of view and notice that the difficulty of solving the code equivalence problem resides in retrieving the pair $(\boldsymbol{S}, \pi)$. Knowing $\boldsymbol{S}$ or π makes the problem easy. One the one hand, if we have $\boldsymbol{S}$ then $\boldsymbol{S}^{-1}\boldsymbol{H}_{\mathrm{pub}}$ and $\boldsymbol{H}$ are two matrices that have the same columns, the mapping between the columns allows to recover π. On the other hand, if we have π, computing the Gaussian elimination on $\boldsymbol{H}^\pi$ gives $\boldsymbol{H}_{\mathrm{pub}}$ and $\boldsymbol{S}$.

On a related note, easier instances with partial knowledge of the private key were proposed in the literature. Table 3 illustrates the existing solutions and the extra information/hints required by each algorithm. In addition, we can remark the different complexity scales for each method, going from polynomial time, when the private polynomial $g(x)$ and a list of $mt + \delta$ points is known, up to sub-exponential or exponential when incorrect and correct points are known. The most realistic scenario in the context of physical attacks is given by the Faulty-Goppa algorithm [KM23] that considers correct and incorrect

[3] π denotes a permutation of the indices in $[\![1; \#\mathcal{L}]\!]$. Such a permutation exists, but is not necessarily the one used in Key generation.

points. It employs a Prange-like decoder which induces the highest complexity of the all the algorithms presented in [KM23]. Conversely, the algorithm proposed requires cubic computation time.

Table 3: Solving the Goppa code equivalence problem with hints.

Article	**Hints**	**Complexity**[4]
[Sen00]	$g(x)$	$\mathcal{O}(n^3 + 2^h n^2 \log n)$
[KM23]	$mt + 1$ correct points α_i	$\mathcal{O}(n^5)$
	$t(m-2) + 1$ correct points α_i and $g(x)$	$\mathcal{O}(n^4)$
	pn incorrect points α_i $(1-p)n > mt + 1$ correct points α_i	$\mathcal{O}\left(n^5 \frac{\binom{n}{mt+1}}{\binom{n(1-p)}{mt+1}}\right)$
This article	$mt + \delta$ correct pairs $(\alpha_i, g(\alpha_i))$	$\mathcal{O}\left(n(mt)^{c-1}\right)$, $c \leq 3$

When setting practical parameters for the KEM (see Table 2) the complexities are computed with respect to the best algorithms for breaking the confidentiality of the scheme. Indeed, breaking the ciphertext has, in general, a complexity cost significantly lower than key recovery attacks. In our case, message recovery attacks boils down to solving the syndrome decoding problem for random linear codes which is estimated using the best information set decoding algorithms [MMT11, BJMM12, BM18, DEEK24].

2.4 Template attacks

Template attacks [CRR03], introduced by Chari *et al.* are the most popular profiled side-channel distinguisher. They assume that the leakages can be modeled as multivariate Gaussian distributions, called templates, and described by a mean vector and a covariance matrix. These templates are built for each class of sensitive information during the profiling stage and exploited later in the matching stage.

The profiling stage consists in estimating the parameters of the multivariate Gaussian distributions for each class. A Gaussian distribution is determined by its two first statistical moments: mean and variance; higher moments are null. To estimate the mean and the variance, the attacker uses a similar device to the one they intend to attack, on which they have full control and can record the physical property of the device. The attacker then runs the target cryptographic operation many times using different values for the private parameter, while using random values for the other inputs so that they do not interfere with the templates creation process, and classifying the resulting traces accordingly. Finally, the templates parameters are computed.

Once all the templates have been created, a power consumption trace t can be associated with a given template, therefore to a given class of the private parameter using the probability density function (PDF) presented in Equation 1.

$$PDF(t|(\mu, \Sigma)) = \frac{1}{\sqrt{(2\pi)^l \times det(\Sigma)}} exp\left(-\frac{1}{2}(t-\mu)^T \Sigma^{-1} (t-\mu)\right) \tag{1}$$

During the matching stage, the attacker records a given number of power consumption traces. All the inputs can be changed except the private parameter to recover, which remains constant. The attacker will then apply the templates on the recorded traces resulting in a probability ranking on the parameter private value. Finally, the attacker picks the most probable guess.

[4] The complexities are given under the assumption that $\boldsymbol{H}_{\text{pub}}$ behaves like a random matrix. For the Support Splitting Algorithm, h is the dimension of the Hull code, the intersection between $\mathscr{C}$ and $\mathscr{C}^{\perp}$.

2.5 Related works: key-recovery attacks on *Classic McEliece*

Although the McEliece [McE78] and Niederreiter [Nie86] cryptosystems have been published respectively in 1978 and 1986, the very large keys prevented them from being implemented in embedded software or hardware for a long time. A first embedded software implementation was published in 2010 [Hey10], but the author opted for reduced security parameters to fit inside the device memory. However, since the NIST PQC standardization process started and larger micro-controllers are available in recent years, several embedded software [RKK20, CC21, SKE+23] and hardware [WSN18, KRF+21, CLHH22, CCD+22] implementations of *Classic McEliece* have been published.

As a consequence of this growing interest in both embedded software and hardware implementations, various physical attacks have been proposed on *Classic McEliece* implementations. We do not consider attacks that recover the message, or short-term secret, from which the session key is derived. Instead, we focus on attacks that recover the private key.

In [PGD+22], a key-recovery fault injection attack is proposed. It is a chosen-ciphertext attack that uses invalid ciphertexts. Additionally, they inject faults during the computation of the error locator polynomial σ and the validity check.

In [GJJ22], a key-recovery attack is performed on both an ARM Cortex-M4 implementation [CC21] and an FPGA implementation [WSN18]. This chosen-ciphertext profiled attack targets the decapsulation algorithm. It relies on a vulnerability in the additive FFT evaluation of the error locator polynomial that allows to recover it through power consumption analysis. They first encrypt every error vector e_i, which has all zero coordinates except the i^{th} one. The obtained ciphertexts are then used in the decapsulation algorithm to compute every possible monic error locator polynomial (i.e. every error locator polynomial of the form $\sigma(x) = x - \alpha_i$, for $i \in [\![0; q]\!]$) while measuring the power consumption. A neural network is then trained by classifying each acquired trace to its corresponding error locator polynomial. Finally, they attack a device by using the previous ciphertexts in the decapsulation algorithm while measuring the power consumption during the additive FFT evaluation. Then, the neural network is used with the acquired traces to infer the corresponding monic error polynomial, and so, the corresponding α. The Goppa polynomial g can then be recovered by factorizing one valid ciphertext and choosing an irreducible factor with weight t. The attack requires n traces, one for each α to recover. The main drawback of this attack is that the ciphertexts used in the attack are created from error vectors with a Hamming weight $\mathsf{wt}(\boldsymbol{e}) = 1$, whereas this should be $\mathsf{wt}(\boldsymbol{e}) = t$ for *Classic McEliece*. A countermeasure could easily detect these malformed ciphertexts and prevent the attack.

In [SCD+23], the described key-recovery attack focuses on the same ARM Cortex-M4 implementation [CC21] of the decapsulation algorithm as the previous attack. It is also a profiled attack. It relies on the fact that, during the loading of the polynomial g coefficients, the power consumption is dependent of the Hamming weight of the coefficients. With this additional information at hand, an exhaustive search through all monic polynomials of degree t for which the coefficients satisfy the Hamming weight constraints can be carried out. The time complexity of this method is exponential.

In the unpublished paper [BCM+23], the authors propose a private key recovery attack on *Classic McEliece* during the public key generation algorithm. The private key is recovered using side-channel leakages during the Gaussian elimination. The attack is simulated in software and requires a very powerful attacker model, where the attacker can perform side-channel measurements during the public key generation.

3 Presentation of the attack

This section first introduces the attacker model as well as the targeted part of the decapsulation algorithm. The course of the attack is then described.

3.1 Attacker model

The attacker model used in this article is the following:

1. The attacker possesses a clone device where the reference implementation of the decapsulation algorithm of *Classic McEliece* can be run as many times as required with various known inputs, and the power consumption of the device can be measured.

2. The attacker has access to the power trace of a *single execution*[5] of the reference implementation of the decapsulation algorithm, using the private key they want to recover, on the device under attack.

3. The attacker has neither control nor knowledge of the ciphertext used as input on the attacked device.

This attacker model is identical to the one considered in [SCD+23]. It is a simpler model than the one considered in [GJJ22] since *no control* over the input ciphertext is required.

3.2 Recovering Hamming weights in the syndrome computation

As shown in Subsection 2.3, in the decapsulation algorithm as described in reference implementation, the ciphertext $\boldsymbol{z}$ is first padded with k zeros to get a new vector $\boldsymbol{v} = (\boldsymbol{z}, 0, \cdots, 0)$. This vector is then multiplied with the parity-check matrix $\boldsymbol{H}_{\text{priv}}$ constructed using $g(x)$, and the permuted support $\mathcal{L}$. Algorithm 5 shows the pseudocode of the reference implementation [ABC+22] of the syndrome computation.

Algorithm 5 Syndrome computation in the reference *Classic McEliece* implementation.

```
Input: The vector $\boldsymbol{v} = (\boldsymbol{z}, 0, \cdots, 0)$, the private key $\mathsf{sk} = (g, \mathcal{L})$
Output: The syndrome $\boldsymbol{s} = \boldsymbol{H}_{\text{priv}_{g^2}} \boldsymbol{v}^{\mathrm{T}}$, where $\boldsymbol{H}_{\text{priv}_{g^2}}$ is generated from $g$ and $\mathcal{L}$
1: for row $\leftarrow 0$ to $2t - 1$ do
2:     $\boldsymbol{s}[\texttt{row}] \leftarrow 0$                                ▷ Initialization
3: for col $\leftarrow 0$ to $n - 1$ do
4:     $b \leftarrow g^{-2}(\mathcal{L}[\texttt{col}])$
5:     for row $\leftarrow 0$ to $2t - 1$ do
6:         $\boldsymbol{s}[\texttt{row}] \leftarrow \boldsymbol{s}[\texttt{row}] + \boldsymbol{v}[\texttt{col}] \times b$        ▷ Syndrome computation
7:         $b \leftarrow b \times \mathcal{L}[\texttt{col}]$                      ▷ Coefficients computation
8: return $\boldsymbol{s}$
```

Recall that $\boldsymbol{H}_{\text{priv}_{g^2}}$ is the parity-check matrix constructed from $g^2(x)$ and $\mathcal{L}$. Hence, Algorithm 5 computes the product $\boldsymbol{H}_{\text{priv}_{g^2}} \boldsymbol{v}^T$ by rebuilding each column i of $\boldsymbol{H}_{\text{priv}_{g^2}}$ and multiplying it with v_i through two nested loops. The first loop, at line 3, iterates over the n columns of the matrix $\boldsymbol{H}_{\text{priv}_{g^2}}$. Once a column i of $\boldsymbol{H}_{\text{priv}_{g^2}}$ has been selected, the coefficient of its first row, i.e. $\boldsymbol{H}_{\text{priv}_{g^2}}[0, i] = g^{-2}(\alpha_i)$, is computed. Then, the second loop, at line 5, iterates over the $2t$ rows of $\boldsymbol{H}_{\text{priv}_{g^2}}[: i]$. For each row j, the value s_j of

[5] This assumes that the signal-to-noise ratio (SNR) is high enough to train a side-channel distinguisher with sufficient accuracy. If this is not the case, averaging can be used to increase the SNR.

the syndrome is updated by adding the product $\boldsymbol{H}_{\text{priv}_{g^2}}[j,i]v_i$ and the coefficient of the following row is computed by multiplying $\boldsymbol{H}_{\text{priv}_{g^2}}[j,i]$ by α_i.

The attack presented in this article targets the implementation of the product $\boldsymbol{H}_{\text{priv}_{g^2}}\boldsymbol{v}^T$ shown on lines 4 to 7 in Algorithm 5. Specifically, $n \times 2t$ Hamming weight values are estimated to build the matrix of weights $\boldsymbol{H}_{\mathsf{wt}}$ shown in Equation 2.

$$\boldsymbol{H}_{\mathsf{wt}} = \begin{pmatrix} \mathsf{wt}(g^{-2}(\alpha_0)) & \dots & \mathsf{wt}(g^{-2}(\alpha_{n-1})) \\ \vdots & \ddots & \vdots \\ \mathsf{wt}(\alpha_0^{2t-1}g^{-2}(\alpha_0)) & \dots & \mathsf{wt}(\alpha_{n-1}^{2t-1}g^{-2}(\alpha_{n-1})) \end{pmatrix} \tag{2}$$

It is important to note that the order of the columns corresponds to the order of the elements in the *permuted* support $\mathcal{L}$.

Given the *Classic McEliece* parameters shown in Table 2, the total number of Hamming weight values ranges from $3488 \times 128 = 446\,464$ to $8192 \times 256 = 20\,996\,096$.

3.3 Private key recovery

The objective of this section is to show how one can exploit the matrix of Hamming weights $\boldsymbol{H}_{\mathsf{wt}}$ to recover both the private Goppa polynomial $g(x)$ and the private permuted support $\mathcal{L}$. We assume here that the attacker can obtain the matrix of weights $\boldsymbol{H}_{\mathsf{wt}}$. We will show in three steps how to recover the whole private key, which is composed of the private Goppa polynomial $g(x)$ and the private permuted support $\mathcal{L}$.

1. Recover $mt+\delta$ pairs of elements $(\alpha_i, g(\alpha_i))_{i\in[\![0;mt+\delta-1]\!]}$. To do so, we will demonstrate through simulations that the Hamming weight of a Vandermonde-like vector $(\alpha_i^j)_{j\in[\![0;d_{m,t}]\!]}$ is an efficient distinguisher if $d_{m,t}$ is large enough.
2. Using only t pairs $(\alpha_i, g(\alpha_i))_{i\in[\![0;t-1]\!]}$, interpolate the private Goppa polynomial $g(x)$.
3. With the knowledge of $g(x)$ and a subset $\mathcal{I}$ of mt points α_i, build the binary expansion of the Vandermonde matrix $\boldsymbol{H}_{g,\mathcal{I}}$ which is a square mt invertible binary matrix. This allows to recover the initial Gaussian elimination matrix and eventually recovers the private permuted support $\mathcal{L}$ completely.

3.3.1 Recovering pairs of elements $(\alpha_i, g(\alpha_i))$

This part corresponds to line 2 of Algorithm 1. The aim is to recover at least $mt+\delta$ pairs $(\alpha_i, g(\alpha_i))_{i\in[\![0;mt+\delta-1]\!]}$ from the matrix of weights $\boldsymbol{H}_{\mathsf{wt}}$. As we shall see in Subsubsection 3.3.2 and Subsection 3.4, only t points are required for polynomial interpolation to recover $g(x)$ but $mt+\delta$ are necessary to recover the original Gaussian elimination matrix and the full permuted support.

In this attack step, the objective is to find a unique pair $(\alpha_i, g(\alpha_i))$ that matches the observed list of Hamming weights $\big(\mathsf{wt}(\alpha^i\beta)\big)_{i\in[\![0;2t-1]\!]}$. This is done by precomputing all the possible $2t$-long lists of Hamming weights $\big(\mathsf{wt}(\alpha^i\beta)\big)_{i\in[\![0;2t-1]\!]}$ for $\alpha \in \mathbb{F}_{2^m}^*$ and $\beta \in \mathbb{F}_{2^m}^*$. This amounts to $(2^m-1)^2$ lists. Considering the largest *Classic McEliece parameters*, this requires the precomputation of $(2^{13}-1)^2 = 67\,092\,481$ Hamming weight lists of length $2t = 256$. As experimentally demonstrated in Section 4, this is easily done even on a laptop. For each Hamming weight list, we store the (α,β) pair which was used to generate it. Later on, given a Hamming weight list $\boldsymbol{H}_{\mathsf{wt}}[:i]$, one can simply compare it with all the precomputed ones and obtain the (α,β) pair, which given the target syndrome computation corresponds to a $(\alpha, g^{-2}(\alpha))$ pair. In order to obtain $g(\alpha)$, we compute $g^2(\alpha) = (g^{-2}(\alpha))^{-1}$, and then $g(\alpha) = \sqrt{g^2(\alpha)} = (g^2(\alpha))^{2^{m-1}}$.

Considering a given Hamming weight list $\boldsymbol{H}_{\mathsf{wt}}[:i]$, three cases can occur when comparing it with the precomputed ones:

Zero pair No (α, β) pair matches the Hamming weight list $\boldsymbol{H}_{wt}[: i]$. It happens if the side-channel Hamming weight estimation is wrong for at least one Hamming weight. With a sufficiently good side-channel estimator, this happens with low probability, as shown experimentally in Subsection 4.1.

Multiple pairs Several (α, β) pairs match the Hamming weight list $\boldsymbol{H}_{\mathsf{wt}}[: i]$. If this happens, $\boldsymbol{H}_{wt}[: i]$ is discarded. We shall see that this situation happens with low probability in general in Conjecture 1 and validate this for the parameters of *Classic McEliece* and of toyeliece in Subsubsection 3.3.1.

Unique pair A unique (α, β) pair matches the Hamming weight list $\boldsymbol{H}_{\mathsf{wt}}[: i]$. With a sufficiently good side-channel estimator, this is the correct $(\alpha, g^{-2}(\alpha))$ pair. The case where this might be another valid but incorrect $(\alpha, g^{-2}(\alpha))$ pair is discussed in Subsubsection 3.3.2, in particular, how to detect invalid pairs.

One possible optimization is to restrict the length of the Hamming weight list from $2t$ to $d_{m,t} < 2t - 1$. This reduces the number of powers in the Vandermonde expansion and makes comparisons faster.

The $d_{m,t}$ parameter is chosen such the aforementioned comparisons recovers $mt + \delta$ correct pairs $(\alpha_i, g(\alpha_i))$ out of n. The soundness of this algorithm is based on the fact that the Vandermonde-type Hamming weights represent a powerful distinguisher for the elements in $\mathbb{F}_{2^m}$.

Hamming weight of the Vandermonde matrix as a distinguisher Although a formal mathematical proof represents a complex and challenging task we provide numerical evidence of the distinguishing capability. Let us define the Hamming weight Vandermonde-like matrix $\beta \boldsymbol{V}_\alpha = \left(\mathsf{wt}(\alpha^i \beta)\right)_{i \in [\![0; d_{m,t}]\!]}$ for $(\alpha, \beta \in \mathbb{F}_{2^m}^* \times \mathbb{F}_{2^m}^*)$ for an integer d.

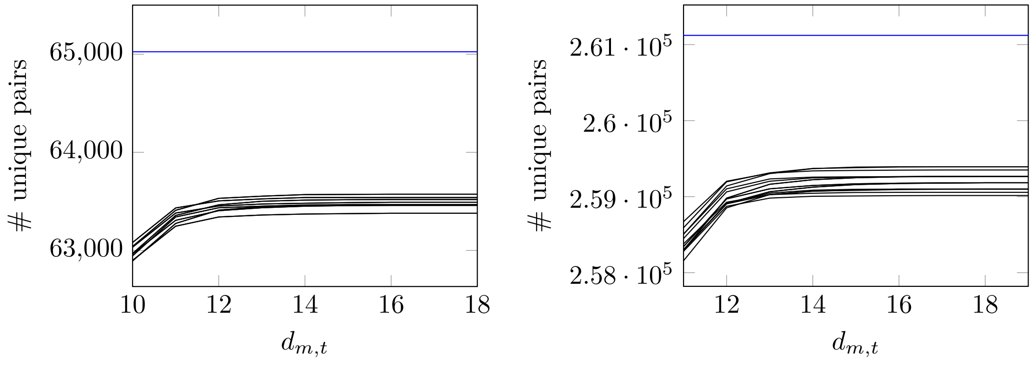
Conjecture 1. *For almost all degree m monic irreducible polynomials $f(x) \in \mathbb{F}_2[x]$ the extension $\mathbb{F}_{2^m}^*$ is such that almost all pairs $(\alpha, \beta) \in \mathbb{F}_{2^m}^* \times \mathbb{F}_{2^m}^*$ can be fully determined using $\beta \boldsymbol{V}_\alpha$ as long as $d_{m,t}$ is sufficiently large.*

This conjecture is made by observing this phenomenon for small binary field $\mathbb{F}_{2^m}$, for $m \leq 9$. While the lists of Hamming weights vary a lot depending on the irreducible polynomial f used to build the extension, their relative frequencies remain very close.

We report the results in Figure 1 for $m = 8$ and $m = 9$. For those parameters, we generated all the irreducible polynomials of degree m and calculated the number of unique pairs while varying the parameter $d_{m,t}$.

We can observe that most pairs can be uniquely identified by the list of their Hamming weights indeed. The number of pairs we can identify depends on the irreducible polynomial used to generate the field. This implies that some polynomials are more sensitive to the proposed distinguisher than others. In particular, over the 9 irreducible polynomials of degree 6 over $\mathbb{F}_2$, one $x^6 + x^3 + 1$ only leads to 43 unique pairs out of $(2^m - 1)^2 = 3969$ while all other polynomials allow to identify more than 89.67 % of the pairs. For $m = 7$, the percentage increases to 99.22 % and is identical for all the 18 irreducible polynomials. For $m = 8$, 2 over the 30 irreducible polynomials lead to distinguish only 167 unique pairs $(x^8 + x^5 + x^4 + x^3 + 1$ and $x^8 + x^7 + x^6 + x^4 + x^2 + x + 1)$ out of 65 025 while all other lead to around 97.47 % of unique pairs. For $m = 9$, for all the 56 irreducible polynomials, the percentage of unique pairs increases to at least 99.19 %.

It is clear from Figure 1 that $d_{m,t}$ is much lower than the maximum length $(2t - 1)$ of the list of Hamming weights. In addition, the number of unique pairs increases with $d_{m,t}$ while converging rather rapidly. For $m = 6$, we only need the first 12 Hamming weights; for $m = 7$, the first 23 are required, but in most cases, we need around 16 to uniquely identify the pair if possible; for $m = 8$, the first 16 are sufficient; and for $m = 9$, we require



(a) $\mathbb{F}_{2^8}$.

(b) $\mathbb{F}_{2^9}$.

Figure 1: Estimation of the number of pairs of elements identified by their list of Hamming weight for different representations of $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^9}$.

18 at maximum. Our computations show a linear trend of $d_{m,t}$ in function of m, at least for small values of m, which is significantly lower than the maximum value.

3.3.2 Recovery of the private polynomial g

This part corresponds to the step 3 of Algorithm 1. Given t distinct pairs $(\alpha_i, g(\alpha_i))_{i\in[\![0;t-1]\!]}$, the private monic Goppa polynomial $g(x)$ of degree t can be reconstructed using Lagrange interpolation.

In [Ber24], D. J. Bernstein details an algorithm (Algorithm 4.1.1 called interpolation with errors following the Reed-Solomon decoding methodology [RS60]) which recovers a polynomial $g(x) \in \mathbb{F}_{2^m}[x]$ with $\deg g = n-2t$ given a vector that matches $(g(\alpha_1), \dots, g(\alpha_n))$ on at least $n-t$ positions. We can use this algorithm to recover the private monic polynomial $g(x)$ in a scenario where some of the pairs $(\alpha_i, g(\alpha_i))_{i\in[\![0;t+2\gamma-1]\!]}$ are incorrect. We need to have at least $t+\gamma$ correct pairs in a set of $t+2\gamma$ pairs, where γ is the number of incorrect pairs. This should be less than the $mt+\delta$ pairs needed to recover the permuted support $\mathcal{L}$.

3.4 Full recovery of the private permuted support $\mathcal{L}$

We require here a subset $\mathcal{I} \subset [\![0; n-1]\!]$ with $\#\mathcal{I} = mt$ and $\mathsf{Rank}(\boldsymbol{H}'[:\mathcal{I}]) = mt$, where $\boldsymbol{H}'$ is the binary extension of the Vandermonde matrix $\boldsymbol{H}$. Notice that this set corresponds to the columns of $\boldsymbol{H}_{\text{pub}}$ for which we have retrieved the pairs $(\alpha_i, g(\alpha_i))$ before. Notice that $\mathsf{Rank}(\boldsymbol{H}'[:\mathcal{I}]) = mt$ implies $\mathsf{Rank}(\boldsymbol{H}_{\text{pub}}[:\mathcal{I}]) = mt$ since $\boldsymbol{H}_{\text{pub}} = \boldsymbol{S}\boldsymbol{H}'$. Under the assumption that $\boldsymbol{H}'$ is random, the probability that $\mathsf{Rank}(\boldsymbol{H}'[\mathcal{I}]) = mt$ equals $\prod_{i=1}^{mt}\left(1-2^{-i}\right)$. For all the *Classic McEliece* parameters, we obtain a probability greater than 0.29. To increase this probability we require having $mt+\delta$ points, more precisely, for a set $\mathcal{I}$ of cardinality $\#\mathcal{I} = mt+\delta$ the probability that $\mathsf{Rank}(\boldsymbol{H}'[:\mathcal{I}]) = mt$ equals $\prod_{i=1+\delta}^{mt+\delta}\left(1-2^{-i}\right)$. Table 4 illustrates how the probability increases as a function of δ.

Table 4: Probability that $\mathsf{Rank}(\boldsymbol{H}'[\mathcal{I}]) = mt$ with $\#\mathcal{I} = mt+\delta$ for $\delta \leq 10$.

$\#\mathcal{I}$	mt	$mt+2$	$mt+4$	$mt+6$	$mt+8$	$mt+10$
$\Pr(\mathsf{Rank}(\boldsymbol{H}'[\mathcal{I}]) = mt)$	0.289	0.578	0.939	0.984	0.996	0.999

Under the assumption that the points $(\alpha_i)_{i\in\mathcal{I}}$ are correctly determined, then matrix $\boldsymbol{S}$ equals $(\boldsymbol{H}^*)^{-1}\boldsymbol{H}_{\text{pub}}[\mathcal{I}]$ where $\boldsymbol{H}^*$ is the binary representation of the Vandermonde matrix $\boldsymbol{H}_{g,\mathcal{I}}$. In addition, the $\boldsymbol{H}'$ matrix obtained at line 3 matches with the matrix obtained in the Key generation as shown on line 4 of Algorithm 2. One can then map the coefficients of $\boldsymbol{H}'$ from $\mathbb{F}_2^m$ to $\mathbb{F}_{2^m}$ to get $\boldsymbol{H}$. Finally, the private permuted support $\mathcal{L}$ is obtained by dividing the second row of $\boldsymbol{H}$ by the first one, essentially computing $\frac{\alpha_i g(\alpha_i)}{g(\alpha_i)}$ for $i \in [\![0; n-1]\!]$.

Algorithm 6 Full recovery of $\mathcal{L}$.

```
Input: $m, t, \boldsymbol{H}_{\mathrm{pub}}, g(x), \mathcal{I} \subset \mathcal{L}$ s.t., $\mathsf{Rank}(\boldsymbol{H}'[:\mathcal{I}]) = mt$
Output: $\mathcal{L}$
Compute $\boldsymbol{H}_{g,\mathcal{I}} = \left(\alpha^j g^{-1}(\alpha)\right)_{j \in [\![0;t-1]\!], \alpha \in \mathcal{I}}$    ▷ Vandermonde
Expand $\boldsymbol{H}_{g,\mathcal{I}}$ into the $mt \times mt$ binary matrix $\boldsymbol{H}^*$    ▷ $\mathbb{F}_{2^m} \to \mathbb{F}_2^m$ mapping
Compute $\boldsymbol{H}' = \boldsymbol{H}^* \boldsymbol{H}_{\mathrm{pub}}[\mathcal{I}]^{-1} \boldsymbol{H}_{\mathrm{pub}}$
Compute $\boldsymbol{H}$ the $t \times n$ matrix from $\boldsymbol{H}'$    ▷ $\mathbb{F}_2^m \to \mathbb{F}_{2^m}$ mapping
Compute $\mathcal{L} = \frac{\boldsymbol{H}[1:]}{\boldsymbol{H}[0:]} \left(= \frac{\alpha g(\alpha)}{g(\alpha)}\right)$
return $\mathcal{L}$
```

3.5 Complexity analysis

To conclude this section, a complexity analysis of the different parts of the attack is given. We assume $m = \Theta(\log_2 n)$ and $t = \Theta(\frac{n}{\log_2 n})$.

Complexity of the recovery of $mt + \delta$ pairs $(\alpha_i, g^{-2}(\alpha_i))$ Searching for a match for a list of Hamming weights $\boldsymbol{H}_{wt}[:i]$ requires $(2^m - 1) \times (2^m - 1)$ comparisons of vectors of length $d_{m,t}$. This is repeated until $mt + \delta$ pairs $(\alpha_i, g^{-2}(\alpha_i))$ are found.

However, this step can be greatly speed-up by precomputing a hash-table of only the unique (α, β) pairs for a given field degree m, using the $d_{m,t}$-long Hamming weight list as a key and the (α, β) as value. Since a lookup in the hash table has constant time complexity $\mathcal{O}(1)$, then the $mt + \delta$ pairs can be recovered in linear time $\mathcal{O}(mt + \delta) = \mathcal{O}(n)$.

Complexity of the recovery of g and $\mathcal{L}$ The recovery of the polynomial g using t pairs $(\alpha, g(\alpha))$ can be done in $\mathcal{O}(t^2)$ using polynomial interpolation.

The complexity of the recovery of the private permuted support $\mathcal{L}$ (Algorithm 6) is dominated by the linear algebra operations. Indeed, we compute the inverse of an mt-binary matrix $(\mathcal{O}((mt)^c))$ [6], we multiply a square mt binary matrix with a $mt \times n$ binary matrix $\left(\mathcal{O}\left(n(mt)^{c-1}\right)\right)$ [7], and finally we compute the component-wise multiplication of two n length vectors in $\mathbb{F}_{2^m}$ $(\mathcal{O}(n^2))$. Consequently, the overall complexity of recovering of g and $\mathcal{L}$ is $\mathcal{O}\left(n(mt)^{c-1}\right) = \mathcal{O}(n^3)$.

4 Experimental results

In this section, the detailed results of the attack against the reference software implementation of *Classic McEliece* submitted to the NIST post-quantum cryptography standardization process [ABC+22] are presented. We consider the parameter sets shown in Table 2.

The target device is an STM32F303RCT6 microcontroller. It is programmed with only the syndrome computation function used in the decapsulation algorithm. The software is compiled with `arm-none-eabi-gcc` version `9.2.1`. To place ourselves in the context of

[6] c is the matrix multiplication constant, i.e., $c \leq 3$, when Strassen algorithm is used $c = 2.807$.

[7] To determine the complexity of the multiplication of a square mt matrix with a $mt \times n$ matrix we compute block square matrix multiplications. There are n/mt blocks, and for each block, we do a multiplication between two square mt matrices. Overall we have $\mathcal{O}\left(\frac{n}{mt}(mt)^c\right)$.

physical attacks on embedded devices, the firmware is compiled using the `-Os` optimization level. This enables all `-O2` optimizations except those that often increase code size.

For the side-channel traces acquisition, the open-source Chipwhisperer platform developed by NewAE [OC14] has been used. All the computation times were measured on a laptop with 16-core CPU running at 4 GHz and 64 GB of RAM.

Subsection 4.1 describes the recovery of the matrix $\boldsymbol{H}_{\mathsf{wt}}$ through a leakage assessment followed by a template attack and Subsection 4.2 exploits those Hamming weights to recover the private key sk which consists of the permuted support $\mathcal{L}$ and the polynomial g.

4.1 $\boldsymbol{H}_{\mathsf{wt}}$ recovery

This part corresponds to the line 1 of Algorithm 1. The objective of this step is to recover the matrix $\boldsymbol{H}_{\mathsf{wt}}$. A profiled template attack, employing a Linear Discriminant Analysis (LDA) for dimensionality reduction, is used to achieve this goal. We recall that the target is the syndrome computation performed in the decapsulation algorithm of the *Classic McEliece* KEM.

4.1.1 Leakage assessment

Prior to the attack, a leakage assessment step is followed. This step confirms that the side-channel trace contains information on the Hamming weights of the coefficient of the matrix $\boldsymbol{H}_{\mathrm{priv}_{g^2}}$. Secondly, this step allows us to find Points of Interest (PoI) for the LDA and template attack phase. The leakage assessment is done with a one-way ANOVA test [BDGN14]. The F statistic is computed considering the set of possible Hamming weight values, for every element h_{ij} in $\boldsymbol{H}_{\mathrm{priv}_{g^2}}$.

We acquire 10 000 traces of the syndrome computation for various random private keys sk and random ciphertext $\boldsymbol{z}$. These are computed from the public key associated with sk and a random, valid message of Hamming weight t.

Results are shown in Figure 2 for the $\mathsf{toyeliece51220}$ parameters. This was selected for readability reasons. We can see different regular patterns in the plot, the first peak close to an F statistic of 2500, corresponds to the leakage of $\mathsf{wt}(g^{-2}(\alpha))$, i.e. the result of line 4 of Algorithm 5. The part without leakage before that corresponds to the evaluation of $g(\alpha)$ and the inversion. Finally, we can see a group of 39 smaller peaks, corresponding to the $2t-1$ remaining power evaluation, i.e. line 7 in Algorithm 5.

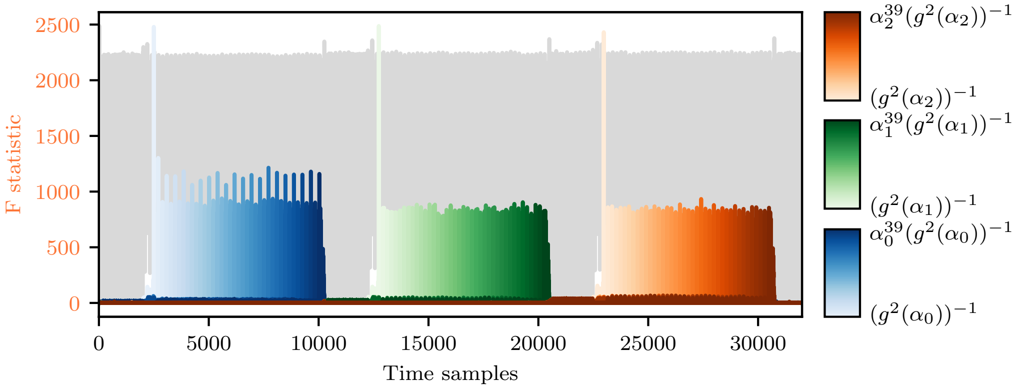


Figure 2: Leakage assessment on the Hamming weight of the coefficients of the matrix $\boldsymbol{H}_{\mathrm{priv}_{g^2}}$ during the computation of $\boldsymbol{H}_{\mathrm{priv}_{g^2}}\boldsymbol{v}^T$ for the $\mathsf{toyeliece51220}$ $(n=512, t=20, m=9)$ parameters. Only α_0, α_1 and α_2 are shown for readability.

4.1.2 Template attack

In this part, the leakage found in the power traces is exploited to recover the Hamming weights. This is done with a standard template attack [CRR03].

Profiling phase Two distinct sets of Hamming weight templates are built:

1. one set corresponding to the evaluation of g in α_i, its squaring and inversion, as shown in line 4 in Algorithm 5. In this case, the Hamming weight templates are built for $\mathsf{wt}(g^{-2}(\alpha_i))$. We refer to the associated training set as $\mathcal{S}_{\text{eval\_sq\_inv}}$. The number of training samples in this set for a single trace is n: it ranges from 512 for toyeliece51220 up to 8192 for mceliece8192128.

2. one set corresponding to the multiplication of $g^{-2}(\alpha_i)$ by successive powers of α_i, as shown in lines 6 and 7 in Algorithm 5. In this case, the Hamming weight templates are built for $\mathsf{wt}(\alpha_i^j g^{-2}(\alpha_i))$. We refer to the associated training set as $\mathcal{S}_{\text{mul\_pow}}$. The number of training samples in this set for a single trace is $2t \times n$: it ranges from 20 480 for toyeliece51220 up to 2 097 152 for mceliece8192128.

Since we are building Hamming weight templates for values in $\mathbb{F}_{2^m}$, $m+1$ templates are necessary. However, some values are under-represented in the training sets. For example, there is only one value of Hamming weight 0 and one value of Hamming weight m. To account for these imbalanced training sets, we use 1000 traces for the profiling phase. Thus, even the most under-represented classes contain enough training samples.

Additionally, we perform a standard dimensionality-reduction step using Linear Discriminant Analysis [SA08] and keep only the first component to build univariate Gaussian templates (μ, σ).

Attack phase After the templates have been built, their accuracy is tested on attack traces. Since the training sets depend only on the field degree m, we only consider the toyeliece51220, mceliece348864 and mceliece8192128 parameter sets. The accuracy of the template distinguishers are given in Table 5. Although the templates we used are univariate, they achieve very good accuracy. The lowest is 0.9684 for mceliece348864 ($m = 12$) when recovering $\mathsf{wt}(\alpha^i g^{-2}(\alpha))$. How this accuracy affects the ability to recover at least $mt + \delta$ pairs $(\alpha, g(\alpha))$ is discussed below.

Table 5: Accuracy of the template distinguishers.

Parameter set	$m = 9$	$m = 12$	$m = 13$
accuracy for $\mathsf{wt}(g^{-2}(\alpha))$	0.9943	0.9940	0.9998
accuracy for $\mathsf{wt}(\alpha^i g^{-2}(\alpha))$	0.9714	0.9684	0.9996

4.2 Key recovery

4.2.1 Hash-table precomputation

Given the parameter sets we consider, the field degree m ranges from 9 to 13. Precomputing the $d_{m,t}$-long lists of Hamming weights for all possible pairs $(\alpha, \beta) \in \mathbb{F}_{2^m}^* \times \mathbb{F}_{2^m}^*$ takes around 2 s for $m = 9$ up to around 9 min for $m = 13$. We make these precomputed hash-tables available on the aforementioned GitLab repository.

4.2.2 Hamming weight list matching

Proposition 1. *In the field proposed in Classic McEliece, for all parameter sets, and also in* `toyeliece`*, the ordered pair* $(\alpha, \beta) \in \mathbb{F}_{2^m}^*$ *generates a list of Hamming weights* $\big(\mathsf{wt}(\alpha^i\beta)\big)_{i\in[\![0;2t-1]\!]}$ *that is unique with high probability.*

Proof. We run the algorithm described in Algorithm 7 to verify empirically that the number of collisions in the list of Hamming weights is small.

Algorithm 7 Collisions in Hamming weight list.

```
Input: $m, t$ parameters from Classic McEliece
Output: count: the number of unique lists of Hamming weights
for $(\alpha, \beta) \in \mathbb{F}_{2^m}^* \times \mathbb{F}_{2^m}^*$ do
    $\mathtt{tab}[\alpha][\beta] = \big(\mathsf{wt}(\alpha^i\beta)\big)_{i\in[\![0;2t-1]\!]}$
$count = 0$
for $(\alpha, \beta) \in \mathbb{F}_{2^m}^* \times \mathbb{F}_{2^m}^*$ do
    if $\forall(\alpha', \beta') \neq (\alpha, \beta), \mathtt{tab}[\alpha][\beta] \neq \mathtt{tab}[\alpha'][\beta']$ then
        $count{+}{+}$
return count
```

We report the number of unique Hamming weight lists in Table 6. One can clearly see that the Hamming weights lists are unique with very high probability.

Table 6: Number of unique lists of Hamming weight.

Parameter set	toyeliece 51220	mceliece 348864	mceliece 460896	mceliece 6688128	mceliece 6960119	mceliece 8192128
# lists	261 121	16 769 025	67 092 481	67 092 481	67 092 481	67 092 481
# unique	259 393	16 676 095	67 084 291	67 084 291	67 084 291	67 084 291
Ratio ($c_{m,t}$)	0.9934	0.9945	0.9999	0.9999	0.9999	0.9999

4.2.3 Attack success rate

From Table 6, we can estimate a lower bound on the probability of obtaining at least $(mt + \delta)$ correct elements α. Suppose that each (α, β) can be correctly detected with the same probability $p_{a,d_{m,t}}$ which depends on the classifier accuracy a and the dimension of the vector $(\alpha^i\beta)$ which was previously defined as $d_{m,t}$. Additionally, we assume that the probabilities in Table 6 are uniformly distributed over the possible Hamming weight lists, i.e., $p_{a,d_{m,t}} = a^{d_{m,t}}c_{m,t}$ where $c_{m,t}$ is the probability of non-collision reported in Table 6 and a is the accuracy of the distinguisher for $\mathsf{wt}(\alpha^i g^{-2}(\alpha))$ (in practice there is one component $g^{-2}(\alpha)$ with a higher accuracy than the other $d_{m,t} - 1$). Hence, the probability of correctly distinguishing at least $mt + \delta$ elements α out of n, that are unique in the weight lists is given in Equation 3.

$$p_{\text{success}} = \sum_{i=mt+\delta}^{n} \binom{n}{i} p_{a,d_{m,t}}^i (1 - p_{a,d_{m,t}})^{n-i} \tag{3}$$

Since we require $\delta \leq 10$ in our attack (as shown in Table 4), we plot in Figure 3 the success probability as a function of the side-channel distinguisher accuracy. Based on our observations on the distinguisher, we have set $d_{m,t} = 22$ for all *Classic McEliece* parameters, and $d_{m,t} = 18$ for toyeliece. For all parameters, we reach 100 % success rate

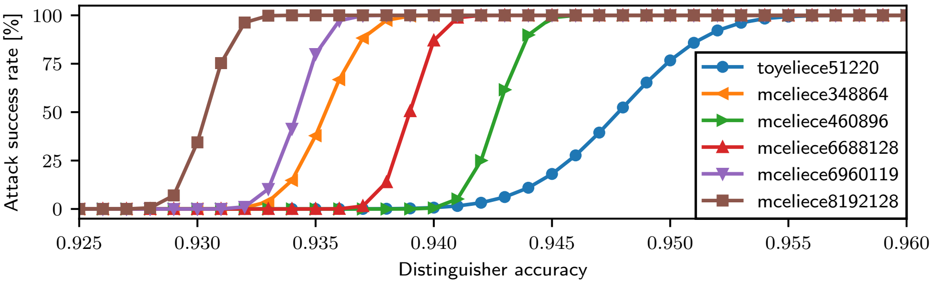


Figure 3: Attack success rate as a function of the distinguisher accuracy.

given the experimental accuracy given in Table 5. Our algorithm is thus able to tolerate classifiers having a lower accuracy, e.g., all cryptographic parameters admit $a \geq 0.945$.

4.3 Overall computation time

Table 7 shows the computation time required for each step of Algorithm 1. We report only the times taken from lines 3 to 6 since the other ones are insignificant in comparison. As discussed in Subsection 3.5 and as expected, linear algebra operations dominate: lines 4 and 5 are the most computationally expensive. Nevertheless, the attack is still very computationally efficient. *Classic McEliece* parameters can be attacked in 0.531 s for mceliece348864 and less than 7 min for mceliece8192128.

Table 7: Average time (in seconds) required for each step of the proposed attack.

Parameter set	line 3	line 4	line 5	line 6	**Total**
toyeliece51220	0.445	0.042	0.028	0.012	0.531
mceliece348864	0.383	1.153	1.017	0.947	3.505
mceliece460896	0.420	0.264	7.575	7.481	15.75
mceliece6688128	0.454	0.521	12.63	12.60	26.22
mceliece6960119	0.485	0.449	13.28	13.00	27.23
mceliece8192128	0.453	0.619	194.1	202.6	397.8

4.4 Single-trace feasibility

Experimental results demonstrated in this section show that, on the chosen ChipWhisperer [OC14] board, the attack can be carried out with a single side-channel trace. One could argue that this fact is board-dependent, although more advanced boards with even less noise have been developed since [JGCS24].

The single-trace argument essentially revolves around the accuracy of the classifier. As experimentally demonstrated, such accuracy is higher than 0.9684 for all *Classic McEliece* parameters considered, as shown in Table 5. This is sufficient to carry out the attack with a single trace, as shown in Figure 3

If the hardware platform has a lower signal-to-noise ratio, then the classifier may not be as accurate, and the single-trace argument may not hold. In this case, averaging over multiple side-channel traces[8] or performing error correction may be a suitable option, but is beyond the scope of this article.

[8] Note that averaging is possible because the attack relies on side-channel leakage from computations that do not involve the ciphertext.

5 Countermeasures

This section briefly discusses possible countermeasures that could be implemented to thwart the proposed attack. These countermeasures are of two types. The first ones are algorithmic countermeasures, which consists in modifying the *Classic McEliece* algorithms shown in Algorithm 2, Algorithm 3 and Algorithm 4. The second ones deal with implementations of these algorithms.

5.1 Algorithmic countermeasures

The following section presents a series of countermeasures based on the modification of certain choices made in the *Classic McEliece* specifications.

Building the extension with a different polynomial f The proposed attack is based on the ability to extract multiple pairs $(\alpha, g(\alpha))$, from their Hamming weights list. However, we have identified instances where some polynomials f used to build the extension lead to Hamming weight lists that do not allow to distinguish unique values in the Vandermonde matrix for specific values of m, specifically 6 and 8. This limitation may be addressed by finding and using such polynomials for $m = 12$ and $m = 13$ to build the extension. However, additional leakages that occur during the evaluation of g and the inversion of $g^2(\alpha)$ could be leveraged to recover g in this case.

Shortening the Vandermonde matrix In the `synd` function of the reference implementation, the ciphertext $\boldsymbol{z}$ is first padded with $n - mt$ zeros. Subsequently, the padded vector is multiplied with the *full* Vandermonde matrix of size $2t \times n$, thus the last $n - mt$ columns of the Vandermonde matrix are multiplied with null coordinates. Another option would be to use a smaller Vandermonde matrix of size $2t \times mt$ and multiply it with $\boldsymbol{z}$. This would provide the attacker with at most mt pairs. However, as shown in Table 4, the probability that $\boldsymbol{H}'[\mathcal{I}]$ is invertible is only 29 % in that case. It prevents an attacker from recovering $\mathcal{L}$ and they must then resort to using the Support Splitting Algorithm [Sen00] or the method proposed by Kirshanova and May [KM23].

5.2 Implementation countermeasures

Classical implementation countermeasures such as masking [ISW03] or shuffling [HOM06] could also be used.

Masking The evaluation of the polynomial g of degree t may be a challenging part to mask, especially for implementations where g is not constant. Hence, masking may require too many resources. Moreover, masking is efficient when sufficient noise is present. Considering the high accuracy of our simple distinguisher, this might not be enough.

Shuffling Applying shuffling on the row accesses in a given column will lead to an inefficient implementation, since it computes the successive power of α, by using the result of the previous loop. Shuffling could be applied to the order of the columns. In that case, the attacker can still recover $(\alpha, g(\alpha))$ pairs and interpolate the polynomial g. However, to compute the change of basis, columns of $\boldsymbol{H}_{\text{pub}}$ are required and this is lost with shuffling. Again, it prevents an attacker from recovering $\mathcal{L}$ and they must then resort to using the Support Splitting Algorithm [Sen00] or another method [KM23].

6 Conclusion

In this paper, we describe a cubic-time full key-recovery attack on *Classic McEliece*. Assuming a simple Hamming weight leakage model, the proposed attack recovers both the Goppa polynomial g and the permuted support $\mathcal{L}$ which make up the private key sk with a new algorithm that runs in cubic time. Experimental results on the ChipWhisperer platform prove that the attack is feasible in practice on the Decapsulation of the reference implementation of *Classic McEliece*.

Future works could improve the attack scenario by requiring a less accurate classifier. Indeed, as highlighted in this article, a rather high accuracy is required for the attack to succeed. One possible research direction relates to the possibility of correcting errors that happen when matching the Hamming weight lists if the side-channel distinguisher is not accurate enough.

The countermeasures put forward should also be thoroughly evaluated, both in terms of protection level and overhead.

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

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.

[AHPT11] Roberto Avanzi, Simon Hoerder, Dan Page, and Michael Tunstall. Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems. *Journal of Cryptograhic Engineering*, 1(4):271–281, 2011.

[BCM+23] Marcus Brinkmann, Chitchanok Chuengsatiansup, Alexander May, Julian Nowakowski, and Yuval Yarom. Leaky McEliece: Secret key recovery from highly erroneous side-channel information. *IACR Cryptol. ePrint Arch.*, page 1536, 2023.

[BDGN14] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Side-channel leakage and trace compression using normalized inter-class variance. In Ruby B. Lee and Weidong Shi, editors, *HASP 2014, Hardware and Architectural Support for Security and Privacy, Minneapolis, MN, USA, June 15, 2014*, pages 7:1–7:9. ACM, 2014.

[Ber15] Elwyn R. Berlekamp. *Algebraic Coding Theory - Revised Edition.* World Scientific Publishing Co., Inc., USA, 2015.

[Ber24] Daniel J. Bernstein. Understanding binary-Goppa decoding. *IACR Communications in Cryptology*, 1(1), 2024.

[BJMM12] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in 2 n/20: How 1 + 1 = 0 improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536. Springer, 2012.

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings*, volume 10786 of *Lecture Notes in Computer Science*, pages 25–46. Springer, 2018.

[BMvT78] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Trans. Inf. Theory*, 24(3):384–386, 1978.

[CC21] Ming-Shing Chen and Tung Chou. Classic McEliece on the ARM Cortex-M4. *IACR TCHES*, 2021(3):125–148, 2021.

[CCD$^+$22] Po-Jen Chen, Tung Chou, Sanjay Deshpande, Norman Lahr, Ruben Niederhagen, Jakub Szefer, and Wen Wang. Complete and improved FPGA implementation of classic McEliece. *IACR TCHES*, 2022(3):71–113, 2022.

[CEvMS16] Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Horizontal and vertical side channel analysis of a McEliece cryptosystem. *IEEE Transactions on Information Forensics and Secuity.*, 11(6):1093–1105, 2016.

[CLHH22] Shaofen Chen, Haiyan Lin, Wenjin Huang, and Yihua Huang. Hardware design and implementation of Classic McEliece post-quantum cryptosystem based on FPGA. In *IEEE High Performance Extreme Computing Conference, HPEC 2022, Waltham, MA, USA, September 19-23, 2022*, pages 1–6. IEEE, 2022.

[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.

[DEEK24] Léo Ducas, Andre Esser, Simona Etinski, and Elena Kirshanova. Asymptotics and improvements of sieving for codes. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 151–180. Springer, 2024.

[FLZG24] Daniel Fallnich, Christian Lanius, Shutao Zhang, and Tobias Gemmeke. Efficient ASIC architecture for low latency classic McEliece decoding. *IACR TCHES*, 2024(2):403–425, 2024.

[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.

[Gop70] Valerii Denisovich Goppa. A new class of linear correcting codes. *Problemy Peredachi Informatsii*, 6(3):24–30, 1970.

[Hey10] Stefan Heyse. Low-reiter: Niederreiter encryption scheme for embedded microcontrollers. In Nicolas Sendrier, editor, *The Third International Workshop on Post-Quantum Cryptography, PQCRYPTO 2010*, pages 165–181. Springer, Berlin, Heidelberg, May 2010.

[HMP10] Stefan Heyse, Amir Moradi, and Christof Paar. Practical power analysis attacks on software implementations of McEliece. In Nicolas Sendrier, editor, *Third International Workshop on Post-Quantum Cryptography*, volume 6061 of *Lecture Notes in Computer Science*, pages 108–125, Darmstadt, Germany, May 2010. Springer.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *ACNS 06International Conference on Applied Cryptography and Network Security*, volume 3989 of *LNCS*, pages 239–252. Springer, Berlin, Heidelberg, June 2006.

[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, Berlin, Heidelberg, August 2003.

[JGCS24] Arpan Jati, Naina Gupta, Anupam Chattopadhyay, and Somitra Kumar Sanadhya. EFFLUX-F2: A high performance hardware security evaluation board. In Romain Wacquez and Naofumi Homma, editors, *COSADE 2024*, volume 14595 of *LNCS*, pages 38–56. Springer, Cham, April 2024.

[KM23] Elena Kirshanova and Alexander May. Breaking Goppa-based McEliece with hints. *Inf. Comput.*, 293:105045, 2023.

[KRF$^+$21] Vatistas Kostalabros, Jordi Ribes-González, Oriol Farràs, Miquel Moretó, and Carles Hernández. HLS-based HW/SW co-design of the post-quantum Classic McEliece cryptosystem. In *31st International Conference on Field-Programmable Logic and Applications, FPL 2021, Dresden, Germany, August 30 - Sept. 3, 2021*, pages 52–59. IEEE, 2021.

[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.

[Lot02] M. Lothaire. *Algebraic Combinatorics on Words*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2002.

[Mas69] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory*, 15:122–127, 1969.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978.

[MMT11] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, Berlin, Heidelberg, December 2011.

[MSSS11] H. Gregor Molter, Marc Stöttinger, Abdulhadi Shoufan, and Falko Strenzke. A simple power analysis attack on a McEliece cryptoprocessor. *Journal of Cryptographic Engineering*, 1(1):29–36, 2011.

[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.

[Pat75] N. Patterson. The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory*, 21(2):203–207, 1975.

[PGD+22] Sabine Pircher, Johannes Geier, Julian Danner, Daniel Mueller-Gritschneder, and Antonia Wachter-Zeh. Key-recovery fault injection attack on the Classic McEliece KEM. In Jean-Christophe Deneuville, editor, *Code-Based Cryptography - 10th International Workshop, CBCrypto 2022, Trondheim, Norway, May 29-30, 2022, Revised Selected Papers*, volume 13839 of *Lecture Notes in Computer Science*, pages 37–61. Springer, 2022.

[RKK20] Johannes Roth, Evangelos G. Karatsiolis, and Juliane Krämer. Classic McEliece implementation with low memory footprint. In Pierre-Yvan Liardet and Nele Mentens, editors, *Smart Card Research and Advanced Applications - 19th International Conference, CARDIS 2020, Virtual Event, November 18-19, 2020, Revised Selected Papers*, volume 12609 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2020.

[RS60] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[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.

[SAB+20] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. 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.

[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.

[Sen00] N. Sendrier. Finding the permutation between equivalent linear codes: the support splitting algorithm. *IEEE Transactions on Information Theory*, 46(4):1193–1203, 2000.

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

[SKE$^+$23] Minjoo Sim, Hyeokdong Kwon, Siwoo Eum, Gyeongju Song, Minwoo Lee, and Hwajeong Seo. Efficient implementation of the Classic McEliece on ARMv8 processors. In Howon Kim and Jonghee M. Youn, editors, *Information Security Applications - 24th International Conference, WISA 2023, Jeju Island, South Korea, August 23-25, 2023, Revised Selected Papers*, volume 14402 of *Lecture Notes in Computer Science*, pages 324–337. Springer, 2023.

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

[WSN18] Wen Wang, Jakub Szefer, and Ruben Niederhagen. FPGA-based niederreiter cryptosystem using binary goppa codes. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 77–98. Springer, Cham, 2018.