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

Key-Recovery Side-Channel Attack on the Berlekamp-Massey Decoding Algorithm in the *Classic McEliece* KEM

Andrei Alexei[1], Marios Omar Choudary[†1] and Vlad-Florin Drăgoi[1,2]

[1] Computer Science Department, Faculty of Automatic Control and Computer Science, National University of Science and Technology POLITEHNICA Bucharest, 060042 Bucharest, Romania,
andrei.alexei@stud.acs.upb.ro,marios.choudary@upb.ro
[2] Faculty of Exact Sciences, Aurel Vlaicu University of Arad, Arad, Romania,
vlad.dragoi@uav.ro

Abstract. In this article, we present a side-channel attack on unprotected implementations of the Berlekamp-Massey (BM) algorithm in both the reference implementation and an embedded implementation of the *Classic McEliece* KEM, where BM is used during decapsulation as part of the decoder implementation. We conduct a chosen cipher-text key recovery attack that exploits the power consumption of the BM, which is highly dependent on the secret Goppa support elements. We exploit the relation between plain-texts of small Hamming weight, secret elements in the Goppa support and power traces using an efficient Template Attack. Our method completely recovers the secret Goppa support for the first parameter set of the *Classic McEliece* KEM using a single attack trace per secret coefficient. The entire support can be recovered in less than 50 seconds. The remaining part of the secret, the Goppa polynomial, is recovered using Kirshanova and May's algorithm ("Breaking Goppa with hints") in less than 1 minute. Our experiments are performed on the reference implementation using the ChipWhisperer-Lite board platform with the ARM Cortex-M4 microcontroller. We also provide further insights into other implementations as well as an overview of possible extensions and limitations of our attack.

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

1 Introduction

The advent of quantum computing poses a significant threat to classical public-key cryptographic systems. Several quantum algorithms enable efficient optimizations for the factorization of large integers and the computation of discrete logarithms on a quantum computer, thus compromising public-key encryption schemes such as RSA and ECC [Gro96, Sho94]. Although large-scale quantum computers capable of executing Shor's algorithm [Sho94] are not yet a reality, rapid advances in quantum technologies suggest their feasibility in the near future, e.g., the new record of 6100 highly reliable qubits [MNB+25]. Such advances spurred the development of post-quantum cryptography (PQC). Despite being quite a new process, post-quantum encryption schemes have existed since 1978. Indeed, Robert J. McEliece proposed the well-known McEliece cryptosystem, the first code-based cryptosystem [McE78], whose security relies on the hardness of decoding linear codes, an NP-complete problem [BMvT78].

†We thank Christ our God for this work. Licensed under Creative Commons License CC-BY 4.0.

Standardization of post-quantum cryptography In response to the quantum threat, the National Institute of Standards and Technology (NIST) initiated a Post-Quantum Cryptography Standardization process in 2016. The set of finalists was announced in two rounds (2024 and 2025): ML-KEM (based on CRYSTALS-Kyber) [oST24a], ML-DSA (based on CRYSTALS-Dilithium) [oST24b], SLH-DSA (based on SPHINCS+) [oST24c], FN-DSA (based on Falcon) [PFH+22], and HQC [AAB+22]. In its last PQC standardization round, the NIST stated that other strong candidates, such as *Classic McEliece* [ABC+22], were left for standardization with other organizations. Concretely, besides NIST's efforts, ISO and the International Electrotechnical Commission (IEC) have released guidance on migration towards PQ-cryptography. They have already standardized stateful hash-based signatures (e.g., LMS, XMSS) in ISO/IEC 14888-4 and have validated, since June 2026, the PQC update for ISO/IEC 18033-2, where *Classic McEliece* and FrodoKEM, two conservative, well-studied key encapsulation mechanisms, were incorporated. *Classic McEliece*, in particular, is valued for its resilience against quantum attacks and its long-standing security record, making it a strong candidate for future industrial adoption.

In its Technical Guideline BSI TR-02102-1 [fSidI25], the BSI has assessed FrodoKEM (specifically FrodoKEM-976 and FrodoKEM-1344) and *Classic McEliece* (with parameter sets mceliece460896, mceliece6688128, and mceliece8192128) as cryptographically suitable for protecting confidential information on a long-term basis. This recommendation is based on a conservative assessment that includes a significant margin of security against future cryptanalytic advances. In particular, *Classic McEliece* is considered *very thoroughly analyzed* (see [fSidI25, Section 2.4.2]).

High-level description of the *Classic McEliece* KEM Albrecht et al. introduced the *Classic McEliece* algorithm [ABC+22] as a classical and robust code-based KEM construction.The key generation is a well-known masking technique of a structured linear code, in this case a binary irreducible Goppa code. The private key is the pair $(g(x), \mathcal{L})$, an irreducible Goppa polynomial $g(x) \in \mathbb{F}_{2^m}[x]$ of degree t and a private Goppa support, which is a subset $\mathcal{L} = \{\alpha_0, \ldots, \alpha_n\}$ of $\mathbb{F}_{2^m}$ in random order. The public key is the standard form of a binary parity-check matrix for the public code denoted $\boldsymbol{H}_{\text{pub}} = (\boldsymbol{I}_{mt} \mid \boldsymbol{T})$, where $\boldsymbol{I}_{mt}$ is the identity matrix. Finding the polynomial $g(x)$ and the elements in $\mathcal{L}$ from the public parity-check matrix boils down to recovering the private key.

In the encapsulation step, a cipher-text is computed, which is a syndrome vector $\boldsymbol{H}_{\text{pub}}\boldsymbol{e}$, where $\boldsymbol{e}$ is a binary vector of Hamming weight t. The session key is the hash of the cipher-text together with the vector $\boldsymbol{e}$. The decapsulation step begins by computing the syndrome vector using the private key. More exactly, $\boldsymbol{s} = \boldsymbol{H}_{\text{priv}}\boldsymbol{v}^T$, where $\boldsymbol{v}$ is the cipher-text and $\boldsymbol{H}_{\text{priv}} = ((\alpha_i^j g^{-2}(\alpha_i))_{0 \le j \le 2t-1})_{0 \le i \le n-1}$ is the private parity-check matrix. Then, a decoding algorithm is used with input $\boldsymbol{s}$ to compute the error-locator polynomial $\sigma(x)$, which admits t roots satisfying $\sigma(\alpha_i) = 0 \leftrightarrow e_i = 1$ where α_i is the element of the Goppa support at index i. Finally, one determines the roots of $\sigma(x)$ and computes the initial vector $\boldsymbol{e}$. The reference implementation as well the optimized implementation propose to use the Berlekamp-Massey (BM) algorithm [Ber15, Mas69] as decoder. We would like to emphasize that using BM is a (very common) implementation choice, however, it is not a KEM specification requirement.

Breaking the confidentiality of the scheme reduces to finding $\boldsymbol{e}$ from $\boldsymbol{v}$, i.e., the adversary recovers the session key. On the other hand, a key recovery attack enables one to compute $g(x)$ and $\mathcal{L}$ and therefore, find all the session keys derived from $g(x), \mathcal{L}$.

Side-channel attacks against *Classic McEliece* The growing interest in practical security has led to numerous side-channel attacks, summarized in Table 1. While such attacks on code-based public-key encryption predate standardization (see [DRBL18]), we focus here on those targeting *Classic McEliece*.

For the **key generation step**, the only known side-channel attack to date is a private key recovery exploiting Gaussian elimination [BCM+25].

Table 1: Power-consumption side-channel attacks and Laser fault injections (grey cells) against *Classic McEliece*.

Step	Algorithm	Key recovery	Message recovery	Cipher-text assumptions
Key gen	Gauss. Elim.	[BCM+25]		$\mathsf{wt}(\boldsymbol{e}) = 1$
Encaps	Matrix-vector mult.		[CCD+21] [CDCG22] [GCCD23] [CGCD25] [BDK+25]	
Decaps	load function	[SCD+23]		$\mathsf{wt}(\boldsymbol{e}) = 1$
	FFT - root		[LNPS20]	
	FFT - root	[GJJ22a]		
	root & check	[PGD+22]		
	synd. comp.	[DCV+24] [VCC+25]		

The **encapsulation step** has received particular scrutiny, as its core operation is matrix–vector multiplication. A laser fault injection attack was introduced in [CCD+21], followed by a single-trace power side-channel attack in [CDCG22], later improved in [GCCD23]. Subsequent works proposed a correlation attack [CGCD25] and enhanced information set decoding with side-channel hints [BDK+25], both successfully recovering the ephemeral key.

Finally, the **decapsulation step** is the most extensively studied phase of this KEM from a side-channel perspective. In [SCD+23], a profiled attack on the load function revealed only part of the private key, as leakage from loading the Goppa polynomial coefficients was insufficient for a full break. In [LNPS20], the session key was recovered via side-channel analysis with multiple FFT observations, and this approach was later extended to full private key recovery in [GJJ22a] under a chosen-ciphertext model exploiting invalid ciphertexts. Similarly, [PGD+22] introduced a fault injection attack targeting error locator polynomial computation and validity checks. Finally, [DCV+24, VCC+25] demonstrated that attacking the syndrome computation with profiled side-channel analysis can reveal the entire private key.

In this article, we propose a side-channel key recovery attack targeting the Berlekamp-Massey decoding algorithm which is used in the decapsulation step, when used in the reference implementation.

Countermeasures Since we target the decapsulation step we shall focus on this part of the *Classic McEliece* KEM. In the past, three vulnerable steps were identified (see Table 2 for power-based side-channel attacks).

Table 2: Key-recovery power-based side-channel attacks on the decapsulation algorithm of the *Classic McEliece* KEM with countermeasures and our attack

Ref	Step	Target	Template	Traces	Attack complexity	Countermeasure
[SCD+23]	LOAD	$g(x)$	$\mathsf{wt}(g_i)$	1	Exponential	SHUFFLE(LOAD)
[DCV+24]	SYND	$g(x), \mathcal{L}$	$\mathsf{wt}(\alpha_i^j g(\alpha_i)^{-2})$	1	Polynomial	SHUFFLE($\boldsymbol{H}_{\text{priv}}\boldsymbol{v}^T$)
This article	BM	$\mathcal{L}$	α_i	n	Polynomial	–
[GJJ22a]	ROOT	$\mathcal{L}$	α_i	n	Polynomial	Check $\sigma(x)$

1. Step LOAD: Load into memory the private polynomial $g(x)$ ([SCD+23]). Here the complexity of the attack is exponential (the authors recover only the Hamming weight of the coefficients of $g(x)$) and thus to recover the key remains unfeasible in practice. However, particular cases, known as weak keys were completely broken. A possible countermeasure is to shuffle the order in which coefficients are loaded. This would increase exponentially the number of possibilities for $g(x)$.

2. Step SYND: Compute the syndrome vector $\boldsymbol{H}_{\text{priv}}\boldsymbol{v}^T$. The attack in [DCV+24] exploits power leakage of the matrix-vector multiplication and manages to recover (using a template attack) the Hamming weight of the private matrix entries. The correct sequence of Hamming weights for each column in $\boldsymbol{H}_{\text{priv}}$ was a distinguisher for the private Goppa support and Goppa polynomial. The authors in [DCV+24] proposed as countermeasure to compute the multiplication using a classical shuffling technique. Also, an FFT implementation would prevent such attacks [CC21].

3. Step ROOT: Compute the roots of $\sigma(x)$. Guo et al. [GJJ22b] attack this step using plain-texts of Hamming weight 1 which map into $\sigma(x) = x - \alpha_i$, hence exploiting the leakage depending on a particular element α_i. To counter such attacks, a check instruction based on the results in [DCCR13] can be implemented. In [DCCR13] it was shown that the error-locator polynomials have few zero coefficients with high probability. Hence, to avoid such attacks one can impose $t - \delta$ non-zero coefficients, where δ is a small constant.

In Listing 1 we provide a possible solution for a secure implementation of the main steps in the decapsulation process based on existing works. For the SECURE-SHUFFLE one can use [VCMKS12], while the FFT solution is provided in [CC21]. All the steps that we intend to securely protect are those that were previously shown to be vulnerable to power-consumption side-channel attacks.

Listing 1: A secure decapsulation implementation of *Classic McEliece* KEM

```
int decrypt(unsigned char *e, const unsigned char *sk, const unsigned
    char *c)
{
    ...
    //load the private key
    for (i = 0; i < SYS_T; i++) {
        g[i] = secure_shuffle_load_gf(sk);
        sk += 2;
    }
    g[ SYS_T ] = 1;
    support_gen(L, sk);

    //compute syndrome
    //secure_shuflle_synd(s, g, L, r); or
    fft_synd(s, g, L, r);

    bm(locator, s);
    //verify that locator is a not valid non-sparse polynomial
    check_valid(locator, delta);

    //the valid locator is evaluated
    root(images, locator, L);
    ...
}
```

Notice that two important steps have not been previously analyzed in the context of side-channel attacks: BM and SUPPORT_GEN. Hence, no countermeasures have been proposed until now. We choose here to focus on BM since it is a generic decoding algorithm (can also be applied to Alternant codes, Generalized Reed-Solomon codes).

Contributions Our article provides several contributions:

- We provide the first key-recovery side-channel attack on *Classic McEliece* KEM reference software implementation targeting the BM algorithm.
- We identify a side-channel leakage in the BM computation when Hamming weight one plain-texts are encrypted. In particular, we demonstrate and conduct a complete full key recovery attack on the MCELIECE348864 parameter set. Experimentally, we validate the attack using the ChipWhisperer-Lite platform with the ARM Cortex-M4 microcontroller.
- We design an efficient template attack which recovers all secret values α_i, using a single attack trace per α.
- We show that possible countermeasures introduce significant and undesirable overhead. Furthermore, we show how to extend our attack to target Hamming weight 2, rendering such countermeasures impractical. Thus, our side-channel attack remains effective against current state-of-the-art countermeasures, requiring deeper investigation for side-channel resilient implementations.

2 Classic McEliece

2.1 Error correcting codes

We will denote integers using lowercase letters, while intervals of integers by $[\![a;b]\!]$. Sets will be denoted by italic upper cases. Matrices and vectors will be denoted using bold uppercase letters, and bold lowercase letters, respectively. The support of a vector will be denoted by $\sup(\boldsymbol{v})$, and represents the set of positions where $v_i \neq 0$. A sub-matrix of a matrix $\boldsymbol{G}$ indexed by a set of rows $\mathcal{I}$ and a set of columns $\mathcal{J}$ is denoted by $\boldsymbol{G}[\mathcal{I}, \mathcal{J}]$. A row or a column of $\boldsymbol{G}$ is denoted by $\boldsymbol{G}[i :]$ and $\boldsymbol{G}[: j]$, respectively.

Finite fields of order q will be denoted by $\mathbb{F}_q$, in particular $\mathbb{F}_{2^m}$ is the extension of $\mathbb{F}_2$ using an irreducible polynomial of degree m. Also, we will consider the mapping $\mathbb{F}_{2^m} \to \mathbb{F}_2^m$ defined by the irreducible polynomial $f(x)$ that generates the extension $\mathbb{F}_{2^m} \sim \mathbb{F}_2/f(x)$.

Moving to coding theory, $\mathscr{C}$ is a linear code of length n and dimension k if $\mathscr{C} \subseteq \mathbb{F}_q^n$ is a vector subspace of dimension k. The usual notation for it is $[n,k]_q$–code. Any vector $\boldsymbol{c} = (c_0,\ldots,c_{n-1}) \in \mathscr{C}$ is called a codeword. Any code $\mathscr{C}$ can be defined using a basis, most commonly called *generator matrix* ($\boldsymbol{G}$) for $\mathscr{C}$, with $\mathsf{Rank}(\boldsymbol{G}) = k$. We say that $\boldsymbol{G}$ is in standard form if $\boldsymbol{G} = (\boldsymbol{I}_{\mathrm{k}} \mid \boldsymbol{V})$. Let $\boldsymbol{v}$ be a vector in $\mathbb{F}_q^n$. The Hamming weight of $\boldsymbol{v}$ is $\mathsf{wt}(\boldsymbol{v}) = \#\{i \in [\![0; n-1]\!] \,|v_i \neq 0\}$, i.e., the cardinality of its support.

Let $\mathscr{C}$ be a $[n,k]_q$ linear code and $\boldsymbol{H}$ be an $(n-k) \times n$ matrix such that for all $\boldsymbol{c} \in \mathscr{C} \Leftrightarrow$ we have $\boldsymbol{H}\boldsymbol{c}^T = 0$. Then $\boldsymbol{H}$ is called a parity-check matrix of $\mathscr{C}$. Any $\boldsymbol{v} \in \mathbb{F}_q^n$ has a syndrome with respect to $\boldsymbol{H}$ which equals $\boldsymbol{s} = \boldsymbol{H}\boldsymbol{v}^T$.

The code family used in the *Classic McEliece* belongs to algebraic codes class and was invented by Valery Goppa [Gop70].

Definition 1 (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

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

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

Hence, a Goppa code is defined by two parameters, $g(x)$ – the Goppa polynomial, and $\mathcal{L}$– Goppa support. A Goppa code is an $[n, k = n - mt]_2$ –code.

2.2 Decoding Goppa codes using Berlekamp-Massey

One can decode up to t errors using the Patterson algorithm [Pat75]. However, since this strategy is difficult to implement in constant-time [Ber24] other decoders are preferred. For example, Berlekamp-Massey [Ber15, Mas69], or the interpolation method [Ber24] are using $\mathscr{G}(g^2, \mathcal{L})$ to decode t errors.

Algorithm 1 Berlekamp-Massey Algorithm

Input: A sequence $s_0, \ldots, s_{2t-1}$
Output: The connection polynomial $C(x)$

$L \leftarrow 0$ ▷ Degree of $C(x)$
$C(x) \leftarrow 1$
$B(x) \leftarrow 1$
$b \leftarrow 1$
$l \leftarrow 1$
for $N = 0$ **to** $2t - 1$ **do**
  $d \leftarrow s_N + \sum_{j=1}^{L} C_j s_{N-j}$ ▷ Compute the discrepancy
  if $d = 0$ **then** ▷ the sequence remains consistent with $C(x)$
    $l \leftarrow l + 1$
  else
    $T(x) \leftarrow C(x)$ ▷ Backup the current polynomial
    $C(x) \leftarrow C(x) - db^{-1}x^l B(x)$ ▷ Update the connection polynomial
    if $2L \leq N$ **then**
      $L \leftarrow N + 1 - L$
      $B \leftarrow T$
      $b \leftarrow d$
      $l \leftarrow 1$
    else
      $l \leftarrow l + 1$
return $C(x)$

Here, we will focus on the BM algorithm 1.The decoding problem refers to recovering the vector $\boldsymbol{e} \in \mathbb{F}_2^n$ from $\boldsymbol{z} = \boldsymbol{c} + \boldsymbol{e}$ assuming $\mathsf{wt}(\boldsymbol{e}) \leq t$ and $\boldsymbol{c} \in \mathscr{G}(g^2, \mathcal{L})$. We skip the details of why $\mathscr{G}(g^2, \mathcal{L}) = \mathscr{G}(g, \mathcal{L})$. First, we need to compute the syndrome vector/polynomial of $\boldsymbol{z}$ with respect to a parity-check matrix of $\mathscr{G}(g^2, \mathcal{L})$, say $\boldsymbol{H}_{g^2,\mathcal{L}}$, i.e., $\boldsymbol{s} = \boldsymbol{H}_{g^2,\mathcal{L}}\boldsymbol{z}, \boldsymbol{s} \in \mathbb{F}_{2^m}^{2t}$, where $\boldsymbol{H}_{g^2,\mathcal{L}} = \boldsymbol{H}_{\text{priv}}$. Since $\boldsymbol{H}_{g^2,\mathcal{L}}\boldsymbol{z} = \boldsymbol{H}_{g^2,\mathcal{L}}\boldsymbol{e}$ this implies that $s_j = \sum_{i \in \text{sup}(\boldsymbol{e})} g^{-2}(\alpha_i)\alpha_i^j$ for all $j \in [\![0; 2t-1]\!]$. The syndrome vector defines a syndrome polynomial

$$S(x) = \sum_{j=0}^{2t-1} s_j x^j = \sum_{j=0}^{2t-1} \left(\sum_{i \in \text{sup}(\boldsymbol{e})} g^{-2}(\alpha_i)\alpha_i^j \right) x^j. \tag{1}$$

From [MS77] we have $(x - \alpha_i)^{-1} = \left(\frac{g^2(\alpha_i) - g^2(x)}{x - \alpha_i} \right) g(\alpha_i)^{-2}$, which implies

$$S(x) = \sum_{i \in \text{sup}(\boldsymbol{e})} \frac{1}{x - \alpha_i} \pmod{g^2(x)} \tag{2}$$

where $S(x) = s_0 + s_1 x + \cdots + s_{2t-1}x^{2t-1}$. This implies, after some simplifications

$$S(x) = \frac{\sigma(x)'}{\sigma(x)} \pmod{g^2(x)} \tag{3}$$

where $\sigma = \prod_{i \in \text{sup}(\boldsymbol{e})}(x - \alpha_i)$ and $\gcd(\sigma, \sigma') = 1$. To compute σ from $S(x)$ one needs to have $\deg(S) \geq 2t$ if the BM algorithm is applied. This is mainly due to the fact that the sequence $s_0, s_1, \ldots s_l$ forms an LFSR sequence with connection polynomial σ.

Shortest LFSR - BM algorithm The BM algorithm computes the *connection polynomial* of an LFSR sequence. Suppose we are given a sequence of feedback coefficients $c_1, \ldots, c_L$ and an initial state $s_0, \ldots s_L$. The LFSR sequence is the output of the linear recursion $s_j = -\sum_{i=1}^{L} c_i s_{j-i}$ for $j \geq L$. The feedback coefficients define the connection polynomial, $C(x) = 1 + c_1 x + \ldots c_L x^L$, with $C(x) = 1$ for $L = 0$. The polynomial C is sequentially constructed using the values s_i as illustrated in 1. It works as follows. Suppose we are given a sequence $\boldsymbol{s} = s_0, \ldots, s_{q-1}$, where q is a positive integer, generated by an LFSR with length L_q. Denote the connection polynomial for this sequence by $C_q(x)$. We want to find the connection polynomial of the sequence $\boldsymbol{s}^* = s_0, \ldots, s_{q-1}, s_q$. The BM algorithm computes the value $d_q = s_q + \sum_{i=1}^{L_q} c_i s_{q-i}$, also known as the discrepancy. If $d_q = 0$, this implies that the LFSR also generates $\boldsymbol{s}^*$, meaning $L_q = L_{q+1}$ and $C_q(x) = C_{q+1}(x)$. Otherwise, the sequence $\boldsymbol{s}^*$ cannot be generated by the same LFSR and a new LFSR must be found. The connection polynomial for $\boldsymbol{s}^*$ can be computed as $C_{q+1}(x) = C_q(x) - d_q d_r^{-1} x^{q-r} C_r(x)$ having $\deg(C_{q+1}) = L_{q+1} = \max\{L_q, q + 1 - L_q\}$.

Application to binary Goppa decoding In [Mas69], equation 17 and later in Theorem 4, it is demonstrated that any LFSR sequence with connection polynomial $C(x)$ satisfies an algebraic equation $s(x) = \frac{D(x)}{C(x)}$ where $D(x) = \sum_{i=0}^{L-1} x^i \sum_{j=0}^{i} c_{i-j} s_j$. In the case of binary Goppa (see (3)) σ is the connection polynomial computed by BM with input $S(x)$.

2.3 The *Classic McEliece* KEM

The parameters proposed by the authors of the *Classic McEliece* for a security level of at least 128-bits are as follows. In the NIST submission a first set of parameters considers $m = 12, n = 3488, t = 64$ (MCELIECE348864). Other parameter sets are used both in the NIST ans ISO submission as follows. The extension $\mathbb{F}_{2^m}$ is $\mathbb{F}_2/f(x)$ where $f(x) = x^{13} + x^4 + x^3 + x + 1$, hence $m = 13$. There are 3 distinct code lengths considered, each with a corresponding decoding parameter: $n = 6688, t = 128$; $n = 6960, t = 119$; $n = 8192, t = 128$.

The KEM is composed of three algorithms, Key Generation 2, Encapsulation 3, and Decapsulation 4.

Algorithm 2 Key generation – *Classic McEliece*

Input: (m, n, t)
Output: The private key $\mathsf{sk} = (g, \mathcal{L})$ and the public key $\mathsf{pk} = \boldsymbol{T}$
1: Generate a random subest $\mathcal{L} = \{\alpha_0, \ldots, \alpha_{n-1}\} \subseteq \mathbb{F}_{2^m}$
2: Generate a random irreducible monic polynomial $g \in \mathbb{F}_{2^m}[x]$ with $\deg(g) = t$
3: Compute $\boldsymbol{H} = \left((\alpha_i^j g^{-1}(\alpha_i))_{j \in [\![0;t-1]\!]}\right)_{i \in [\![0;n-1]\!]}$
4: Transform $\boldsymbol{H}$ a binary matrix $\boldsymbol{H}'$
5: Compute the standard-form of $\boldsymbol{H}'$: $\boldsymbol{H}_{\text{pub}} = (\boldsymbol{I}_{mt} \mid \boldsymbol{T})$
6: **return** $\mathsf{sk} = (g, \mathcal{L})$ and $\mathsf{pk} = \boldsymbol{T}$

2.4 Breaking Goppa codes (with hints)

Breaking the *Classic McEliece* key resumes to solving the Goppa code equivalence problem. Indeed, given $\boldsymbol{H}_{\text{pub}}$ it is considered difficult to compute $g(x), \mathcal{L}$. First of all, we do not

Algorithm 3 Encapsulation – *Classic McEliece*

Input: $\mathsf{pk} = \boldsymbol{T}$
Output: A cipher-text $\boldsymbol{z}$ and a 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

Algorithm 4 Decapsulation – *Classic McEliece*

Input: $\boldsymbol{z}$ and $\mathsf{sk} = (g, \mathcal{L})$
Output: The session key K
1: Pad $\boldsymbol{z}$ with $n - mt$ zeros to obtain: $\boldsymbol{v} = (\boldsymbol{z}, 0, \ldots, 0)$
2: Compute $\boldsymbol{H}_{\text{priv}} = \left((\alpha_i^j g^{-2}(\alpha_i))_{j \in [\![0;2t-1]\!]} \right)_{i \in [\![0;n-1]\!]}$
3: Compute $\boldsymbol{s} = \boldsymbol{H}_{\text{priv}} \boldsymbol{v}^T$
4: Compute $\sigma(x)$ using the BM algorithm
5: Recover $\boldsymbol{e}$ from σ
6: Compute $K = \mathsf{hash}(1|\boldsymbol{e}|\boldsymbol{z})$
7: **return** K

know which elements from $\mathbb{F}_{2^m}$ are in $\mathcal{L}$ and even in the case $\mathcal{L} = \mathbb{F}_{2^m}$ we do not know the order of the elements of $\mathbb{F}_{2^m}$ in $\mathcal{L}$. Secondly, there are roughly as many Goppa codes as possible polynomials g, number that scales asymptotically as $O(2^{mt}/mt)$ [LS01]. Hence, a possible solution would be to 1) enumerate all possible g and 2) find $\mathcal{L}$ with the knowledge of g using the Support Splitting Algorithm [Sen00].

Obtaining additional information about the private key $(g(x), \mathcal{L})$ significantly reduces the complexity of the key recovery problem. In Table 3 we enumerate existing solutions while emphasizing known types of extra knowledge about the key. As we shall see, here we will target the Goppa elements, hence our attack will belong to Type II class.

Table 3: Solutions for computing the private key of the *Classic McEliece* KEM using partial knowledge of the secret ($g(x)$ or $\mathcal{S} \subseteq \mathcal{L}$), where h is the dimension of $\mathscr{C} \cap \mathscr{C}^{\perp}$.

Type	Knowledge	Complexity	Reference
I. Goppa polynomial	$g(x)$	$\mathcal{O}(n^3 + 2^h n^2 \log n)$	[Sen00]
II. Goppa points	$\#\mathcal{S} = mt + 1$ $\mathcal{S} = \mathcal{L}$	$\mathcal{O}(n^5)$ $\mathcal{O}(n^3)$	[KM23, Thm. 4] [KM23, Thm. 1]
III. Combined	$\#\mathcal{S} = tm + 1$ and $g(x)$ $\#\mathcal{S} = tm + \delta$ and $\{g(\alpha_i) \mid \alpha_i \in \mathcal{S}\}$	$\mathcal{O}(n^4)$ $\mathcal{O}\left(n^3\right)$	[KM23, Thm. 3] [DCV+24]

3 Attack methodology

Before explaining each step of our attack phase we shall give a general overview. Figure 1 illustrates all of the main steps.

1. We generate a plain-text of weight 1, $\boldsymbol{e} \in \mathbb{F}_2^n$ with $\boldsymbol{e} = (0, \ldots, 0, 1, 0, \ldots, 0)$ $(e_i = 1)$. Then we compute the cipher-text $\boldsymbol{z} = \boldsymbol{H}_{\text{pub}} \boldsymbol{e}$.

2. Send $\boldsymbol{z}$ to the Decryption step (we use the reference implementation) on a ChipWhisperer-Lite development board, the attack device. We target the BM decoder.

3. We isolate and record, using our oscilloscope, the power consumption of the BM.

4. The recorded traces are then processed: we compute statistic values (mean vectors, etc.) for a Template Attack to create a power distribution for all the possible values $\alpha \in \mathbb{F}_{2^m}$. During the attack step, we use an efficient linear discriminant to identify the secret value α used with a given attack trace.

5. We deduce that on position i the Goppa support $\mathcal{L}$ equals α. Finally, we repeat this for all possible plain-texts to obtain the full key $\mathcal{L}$.

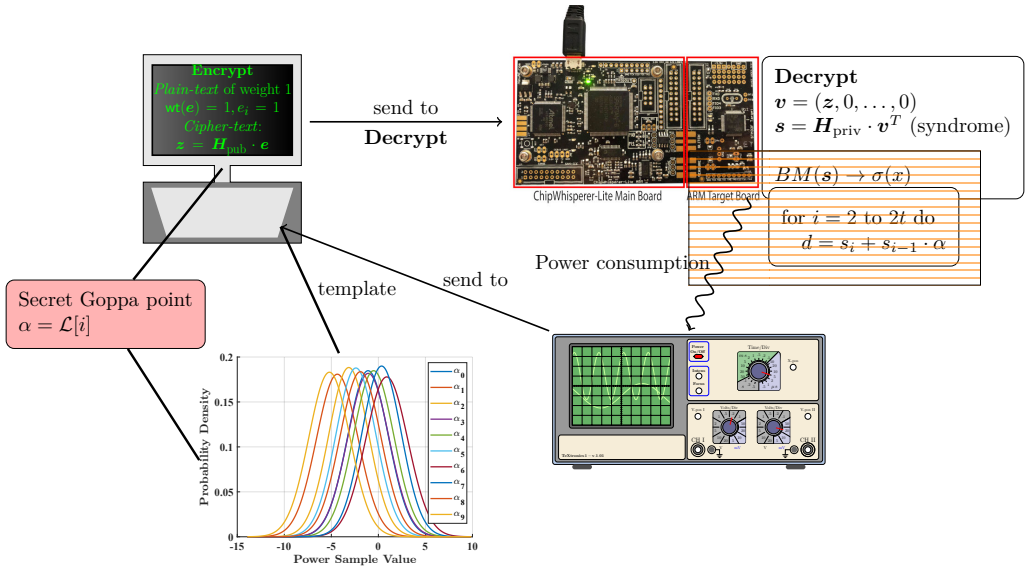


Figure 1: Attack steps: 1) Encrypt plain-text of weight 1, 2) Send the corresponding cipher-text to decryption device, 3) Record power consumption of the BM computation, 4) Compute the statistic profile of the recorder trace and discriminate w.r.t. the trained profiles 5) Deduce the secret value α and repeat this process for all weight 1 plain-texts.

3.1 Chosen cipher-text model

We chose plain-texts such that $\mathsf{wt}(\boldsymbol{e}) = 1$, thus isolating each unknown element $\alpha_i \in \mathcal{L}$. The corresponding syndrome is $\boldsymbol{s} = \boldsymbol{H}_{\text{priv}}[, i] = (g^{-2}(\alpha_i), \ldots, g^{-2}(\alpha_i)\alpha_i^{2t-1})$. Therefore, the computations in the Berlekamp-Massey algorithm will mainly depend on α_i. In general, the two steps that carry almost the entire computational load in the BM algorithm illustrated in Listing 1 are STEP 7, the discrepancy computation, and STEP 12, the connection polynomial updating. However, in our scenario of weight 1 plain-texts, STEP 12 is executed only two times, while STEP 7 is executed $2t$ times (see Table 4). Recall that the discrepancy is computed at each step using the formula

$$d = s_N + \sum_{j=1}^{L} C_j s_{N-j}. \tag{4}$$

Hence, by computing the value d one verifies whether the connection polynomial $C(x)$ generates the sequence $s_0, s_1, \ldots, s_N$ (see details in Table 4). Notice that except for the first two iteration steps, the discrepancy equals zero.

For $\mathsf{wt}(\boldsymbol{e}) = 1$ we have that $\sigma(x) = x - \alpha_i$ hence the syndrome satisfies the recurrence $s_{k+1} + \alpha_i s_k = 0, \forall k \geq 0$, this being the invariant in the BM decoder. Once the connection polynomial $C(x) = X + \alpha_i$ is found every discrepancy will be equal to zero,

$$d = s_N + \alpha_i s_{N-1} = 0. \quad \text{(Step 7 of Alg. 1)} \tag{5}$$

Table 4: Computations in the Berlekamp-Massey algorithm for a plain-text with $\mathsf{wt}(\boldsymbol{e}) = 1$ for the *Classic McEliece* KEM

Iter. (N)	syndrome components	Compute d – STEP 7	Update C – STEP 12	b
Init			$[1]$	1
$N=0$	$[g(\alpha_i)^{-2}]$	$d = s_0 = g(\alpha_i)^{-2}$	$[1, g(\alpha_i)^{-2}]$	$g(\alpha_i)^{-2}$
$N=1$	$[g(\alpha_i)^{-2}, g(\alpha_i)^{-2}\alpha_i]$	$s_1 + s_0 C_1 = g(\alpha_i)^{-2}\alpha_i + g(\alpha_i)^{-2}g(\alpha_i)^{-2}$	$[1, g(\alpha_i)^{-2} + d/b] = [1, \boldsymbol{\alpha_i}]$	$g(\alpha_i)^{-2}$
$N=2$	$[g(\alpha_i)^{-2}\alpha, g(\alpha_i)^{-2}\alpha_i^2]$	$s_2 + s_1 C_1 = g(\alpha_i)^{-2}\alpha_i^2 + g(\alpha_i)^{-2}\alpha_i\boldsymbol{\alpha_i} = 0$	$[1, \boldsymbol{\alpha_i}]$	$g(\alpha_i)^{-2}$
$N=3$	$[g(\alpha_i)^{-2}\alpha^2, g(\alpha_i)^{-2}\alpha_i^3]$	$s_3 + s_2 C_1 = g(\alpha_i)^{-2}\alpha_i^3 + g(\alpha_i)^{-2}\alpha_i^2\boldsymbol{\alpha_i} = 0$	$[1, \boldsymbol{\alpha_i}]$	$g(\alpha_i)^{-2}$
⋮	⋮	⋮	⋮	⋮
$N=i$	$[g(\alpha_i)^{-2}\alpha^{i-1}, g(\alpha_i)^{-2}\alpha_i^i]$	$s_i + s_{i-1} C_1 = g(\alpha_i)^{-2}\alpha_i^i + g(\alpha_i)^{-2}\alpha_i^{i-1}\boldsymbol{\alpha_i} = 0$	$[1, \boldsymbol{\alpha_i}]$	$g(\alpha_i)^{-2}$
⋮	⋮	⋮	⋮	⋮

From a side-channel perspective, (5) becomes interesting. To compute the discrepancy we first multiply α_i with a value from $\mathbb{F}_{2^m}$ and then we add the result to another value in $\mathbb{F}_{2^m}$ (Step 7 in BM - illustrated in Table 4). This is repeated $2t-2$ times. The only apparent constant in these computations is α_i. In addition, for weight 1 plain-text Step 7 in BM carries the most significant computation load.

3.2 Leakage Model

Given the process of the BM algorithm described above, our targets for a side-channel attack are the secret coefficients α_i of the private Goppa support $\mathcal{L} = \{\alpha_0, \ldots, \alpha_{n-1}\}$ with $\alpha_i \in \mathbb{F}_{2^m}$, with $m = 12$ in our implementation.

Many side-channel attacks consider a chosen-cipher-text attack scenario, in which the target variable providing leakage depends on two values: a fixed but *secret* value (e.g. a key word) and a variable but *known* value (e.g. a plain-text word). However, in our context, we are faced with only a *fixed secret* value (the α_i secret coefficients) that is processed during the BM algorithm. Therefore, the leakage function in our context can generally be modeled as

$$l(\alpha_i) = lm(f(\alpha_i)) + \mathcal{N}(0, \sigma), \tag{6}$$

where lm represents some arbitrary leakage model (e.g. Hamming Weight [BCO04] or full model, such as in Template Attacks [CK14, CK17]) for the function $f(\alpha)$ operating over the secret value α (e.g. the multiplication between α and some unknown syndrome component) and the Gaussian distribution $\mathcal{N}(0, \sigma)$ models the noise affecting the traces.

Recall that the BM algorithm requires many instructions (see Table 4) that operate on the same secret coefficient α_i and also note that during a side-channel attack we may obtain many leakage samples for each such instruction. Therefore, during the execution of the BM algorithm, for a given α_i we obtain leakage traces $\mathbf{x}_i \in \mathbb{R}^{n_s}$, $\mathbf{x}_i = \{\mathbf{x}_{i1}, \mathbf{x}_{i2}, \ldots, \mathbf{x}_{in_s}\}$, where each leakage sample $\mathbf{x}_{ij}$ in the trace can be modeled as[1]:

$$\mathbf{x}_{ij} = lm(f_j(\alpha_i)) + \mathcal{N}(0, \sigma_j), \quad 1 \le j \le n_s. \tag{7}$$

3.3 F-test

To evaluate whether the Berlekamp-Massey algorithm leaks information about the secret support values α_i, we used the F-test [Cho15, CK17] over the Berlekamp-Massey execution.

As shown by Choudary and Kuhn, the F-test (aka F-score) is a statistical test that can be used to reject, at any desired significance level, the hypothesis that the sample mean values of different traces at a selected sample point t are equal, therefore providing a good

[1] For generality, we assume that for each sample $\mathbf{x}_{ij}$ the corresponding leakage function f_j can be different, e.g. due to manipulation of different bits, using different hardware components, etc.

indication in which samples the variation is more related to the desired target value. The F-test for a given sample point t is typically computed as:

$$F(t) = \frac{\left(n_{\mathrm{p}} \sum_{\alpha \in \mathcal{S}} (\bar{\mathbf{x}}_\alpha(t) - \bar{\mathbf{x}}(t))^2 \right) / (|\mathcal{S}| - 1)}{\left(\sum_{\alpha \in \mathcal{S}} \sum_{i=1}^{n_{\mathrm{p}}} (\mathbf{x}_{\alpha i}(t) - \bar{\mathbf{x}}_\alpha(t))^2 \right) / (|\mathcal{S}|(n_{\mathrm{p}} - 1))}, \tag{8}$$

where n_{p} is the number of traces recorded for each particular secret value α, $\bar{\mathbf{x}}_\alpha$ is the mean vector for the traces corresponding to α, $\bar{\mathbf{x}}$ is the overall mean across all mean vectors and $\mathcal{S}$ represents the secret support with all values α_i, of size $|\mathcal{S}|$. We can use the F-test to reject the null hypothesis of equal means at a significance level A if

$$F(t) > F_{(|\mathcal{S}|-1),(|\mathcal{S}|(n_{\mathrm{p}}-1))}(A),$$

where $F_{(|\mathcal{S}|-1),(|\mathcal{S}|(n_{\mathrm{p}}-1))}(A)$ is the upper (100A)th percentile of the F-distribution with $(|\mathcal{S}| - 1)$ and $(|\mathcal{S}|(n_{\mathrm{p}} - 1))$ degrees of freedom.

The F-test provides the same utility as the Test Vector Leakage Assessment (TVLA) [GGJR+11]: it detects whether there is a clear side-channel leakage about some target value and it shows which samples provide most leakage across a side-channel trace. In our experiments we have tried both TVLA and F-test, but we decided on the F-test because the traces that we acquire for the Template Attacks described below can be directly used with the F-test, while for TVLA we need a different acquisition[2].

3.4 Classification using Template Attacks

Recall that our context is that of a chosen cipher-text attack, in which we additionally allow the attacker to use side-channel traces to determine the secret coefficients α_i of the private Goppa support $\mathcal{L} = \{\alpha_0, \ldots, \alpha_{n-1}\}$ with $\alpha_i \in \mathbb{F}_{2^m}$.

In this scenario, considering that the side-channel leakage only depends on the secret coefficients α_i processed during the Berlekamp-Massey algorithm, the most suitable side-channel attack is based on splitting the attack into two parts: a profiling step and an attack (or classification) step. These are *profiled attacks*, considered among the most powerful kind of side-channel attacks. To implement profiled side-channel attacks there are two main approaches: multivariate statistics (aka Template Attacks [CRR02, CK17]) and Machine Learning [HGDM+11, HGG20, PPM+23]. The former is known for being very efficient in contexts of linear dependencies and relatively fixed execution environments, while the latter is better suited for non-linear dependencies as well as contexts with jitter, desynchronisation or other possible countermeasures.

Given the well-controlled setup that we use, as we want to explore a worst-case scenario, we have opted for the use of optimized Template Attacks, based on the efficient implementations of Choudary and Kuhn [CK14, CK17]. In short, during the profiling phase, for each $\alpha \in \mathbb{F}_{2^m}$, we obtain n_{p} leakage traces $\mathbf{x}_i^\alpha$, $i = 1, \ldots, n_{\mathrm{p}}$, and we compute the template parameters $\bar{\mathbf{x}}_\alpha \in \mathbb{R}^{n_{\mathrm{s}}}$ and $\mathbf{S}_\alpha \in \mathbb{R}^{n_{\mathrm{s}} \times n_{\mathrm{s}}}$ for each α. Then, in the *attack* phase, we obtain n_{a} leakage vectors $\mathbf{x}_i \in \mathbb{R}^{n_{\mathrm{s}}}$ from the attacked device corresponding to the processing of the secret value $\alpha^\star \in \mathbb{F}_{2^m}$ and we compute a *discriminant score* D for each candidate α. This allows us to sort the candidate values α in decreasing order of likelihood.

To improve the efficiency of our Template Attack, we apply a two-stage pre-processing step to the traces during profiling and attack: firstly, we select only a subset of samples

[2]For TVLA we required about 1000 traces for a fixed value α_f and 1000 traces for random values α_r, while for the Template Attacks we acquired less traces for each α_i, but for all the α_i in the support. Hence, we could directly use these traces with the F-test.

around the peaks given by the F-test; secondly, we apply Linear Discriminant Analysis (LDA) [SA08] to project the traces into a space where the differences between samples corresponding to different candidates α_i are maximised. This allows us also to use a very efficient *linear discriminant* [CK17], which avoids the use of covariance matrices in the attack step, hence improving the speed and performance of attacks considerably.

4 Experimental platform

4.1 Hardware Setup

For the implementation of the attack, we selected the ChipWhisperer-Lite development board from NewAE Technology as the target platform, featuring a STM32F303RCT6 32-bit ARM Cortex-M4 microcontroller . The firmware was compiled using the `arm-none-eabi-gcc` toolchain (version 6.3.1) with optimization level `-O3`, chosen to emulate practical deployment conditions. Additionally, we set the Cortex-M4's clock frequency to 45 MHz.

Table 5: Oscilloscope configurations used in our experiments.

Config	Sampling Freq.	Downsampling Mode	Downsampling Ratio	Raw Samples	Selected Samples
A	1.25 GHz	Averaging	14	50.000	990
B	1.25 GHz	Simple Decimation	80	3938	3938
C	1.25 GHz	Averaging	14	55.000	5300 / 4585

We performed the power trace acquisition using a PicoScope 6404D digital oscilloscope equipped with a differential active probe . We used a sampling frequency of 1.25 GHz with different downsampling configurations, obtaining various lengths for the final power traces, as shown in Table 5. We employed these configurations across the experiments we present in the following sections.

4.2 Experimental Leakage Assessment

To assess whether the Berlekamp–Massey algorithm leaks information about the secret support values in our Hamming Weight one scenario, we employed the F-test as described in Section 3.3. All of our methods were implemented and tested on the first parameter set of the *Classic McEliece* KEM, namely $n = 3488, m = 12, t = 64$.

Capture Window I (Config A, Table 5): To identify the most informative capture window, we first applied the F-test across the execution of the BM, but limiting the computation of the discrepancy (lines 12–20 in Listing 2) only for the iterations $N = 2$ to $N = 6$ and using only a subset of 16 values $\alpha \in \mathbb{F}_{2^{12}}$. The resulting F-statistic plot is shown in Figure 2 (left), where 5 distinct peaks are clearly visible, each corresponding to a discrepancy computation step for $N = 2$ to $N = 6$.

However, we observe that the leakage is mostly due to the first execution of line 17 from Listing 2, when for $i = 0$ the operand $C[i]$ holds the value of the target α_i, which is then used in the GF multiplication shown in Listing 3. This motivates narrowing the capture window to the first discrepancy computation alone.

Capture Window II (Config B, Table 5)): Based on the previous observation, we narrowed the capture window to the first execution of line 17 in Listing 2, capturing 63 iterations of the BM loop out of the possible 126 iterations[3]. With this refined window, we

[3]Note that the first two iterations of the discrepancy do not contain our target value α.

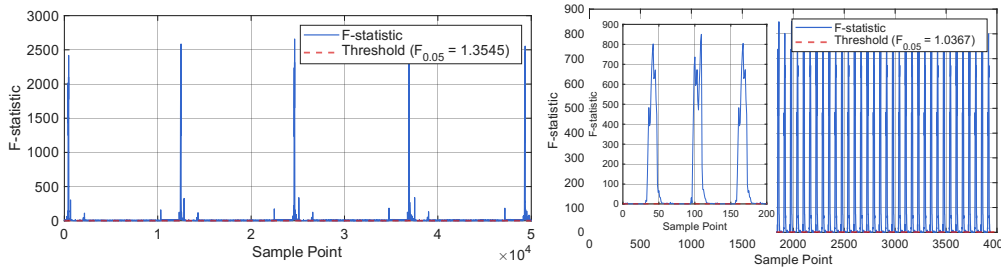


Figure 2: Point-wise F-test across outer BM loop (left) and Focused Point-wise F-test over Berlekamp–Massey (right).

collected a dataset of 400 power traces per value $\alpha \in \mathbb{F}_{2^{12}}$, which was then used uniformly across all experimental phases — namely, the F-test, template construction, and the attack itself.

The resulting F-statistic plot for the refined window is shown in Figure 2 (right), where 63 distinct peaks of high leakage are clearly visible. These peaks correspond to the repeated operation in which a targeted value α_i is multiplied with a syndrome component, as detailed in Table 4. We show in Listing 3, the carry-less multiplication used by BM in the reference code, where the operand `t0` holding the target value α_i is multiplied GFBITS = 12 times, hence leading to the high leakage observed.

Listing 2: BM algorithm

```
void bm(gf *out, gf *s)
{
    ...

    for (i = 0; i < SYS_T+1; i++)
    {
        C[i] = B[i] = 0;
    }

    B[1] = C[0] = 1;

    for (N = 0; N < 2 * SYS_T; N++)
    {
        d = 0; // discrepancy

        for (i = 0; i <= min(N, SYS_T); i++)
            d ^= gf_mul(C[i], s[N - i]);

        ...
    }

    ...
}
```

Listing 3: GF multiplication

```
gf gf_mul(gf in0, gf in1)
{
  int i;

  uint32_t tmp;
  uint32_t t0;
  uint32_t t1;
  uint32_t t;

  t0 = in0;
  t1 = in1;

  tmp = t0 * (t1 & 1);

  for (i = 1; i < GFBITS; i++)
    tmp ^= (t0 * (t1 & (1 << i)));

    ...
}
```

4.3 Template Construction

Based on the findings from Section 4.2, we constructed two distinct datasets of power traces, each corresponding to one of the capture windows identified in the BM implementation. For both datasets, we captured 400 power traces for every possible secret value $\alpha \in \mathbb{F}_{2^{12}}$ $(1,638,400$ traces in total). To ensure variability, we randomized the position of each α_i within a support vector $\mathcal{L}$ across captures using different keys.

We subsequently partitioned the collected traces into two disjoint subsets: a training set used during the profiling phase, and an attack set reserved for evaluation. For the

Capture Window I, we allocated 360 traces per class for template construction, while for the *Capture Window II* we employed 385 profiling traces per class.

To reduce the dimensionality of the power traces while retaining the most discriminative features, we used Linear Discriminant Analysis (LDA). Based on exploratory parameter search, we selected 10 LDA components for the *Capture Window I* and 50 components for *Capture Window II*. Note that the original traces had 990 and 3938 samples for *Capture Window I* and *Capture Window II* settings, respectively (cf. Table 5).

5 Attack implementation and evaluation

To evaluate the success of our attacks in the Hamming weight one scenario for the $m = 12$ parameter set, we used the *empirical guessing entropy* [RPC22] (GE), i.e. the average *rank* [SMY09]. The rank is defined as the position of the correct secret value (e.g. a key byte) in the list of possible values, sorted using information from the side-channel attack; alternatively, the rank is the number of candidate values more likely than the correct value plus one. The lower the rank, the better the attack performs.

To obtain the average rank (empirical guessing entropy), we performed the attack N times (usually $N = 20$) for each secret value $\alpha \in \mathbb{F}_{2^{12}}$, using a different subset of attack traces in each repetition (in total $N \cdot 2^m$ attacks for each set of parameters). Let $GE(\alpha)$ be this empirical guessing entropy for each secret value α, $\mathbf{X}_{\alpha_i}$ be the set of attack traces corresponding to a particular α_i and $\text{rank}(\alpha_i)$ be the rank of α_i after the Template Attack. Then:

$$GE(\alpha) = \frac{\sum_{i=1}^{10} \text{rank}(\alpha_i) \text{ in } \mathbf{X}_{\alpha_i}}{N}. \tag{9}$$

Next, we will show the 25th, 50th (median) and 75th quartiles of GE, when computed across all possible values of the secret value α.

5.1 Guessing entropy results

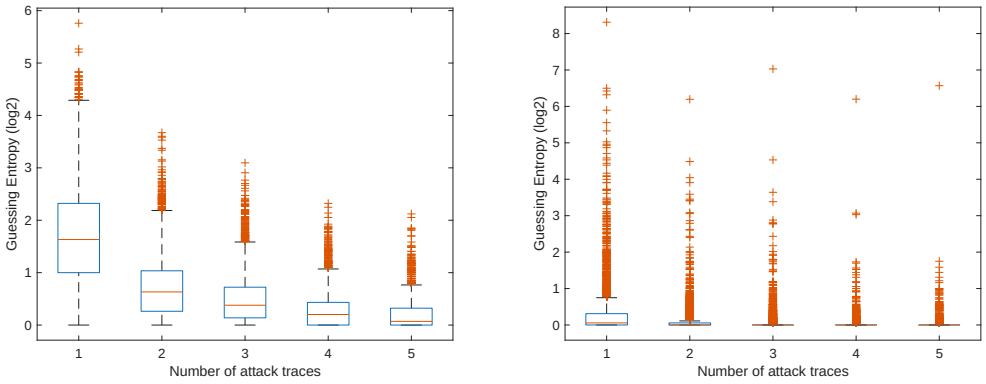


Figure 3: Box plots showing median and 25th/75th percentile of Guessing Entropy for $n_a = 1 \ldots 5$ attack traces on *Capture Window I* (left) and *Capture Window II* (right) settings.

We show the attack performance for both *Capture Window I* and *Capture Window II* settings in Figure 3 and Table 6 (for detailed values). The improved configuration, *Capture Window II*, yields significantly better performance. In particular, with a single attack trace the median of GE is 0.05., showing that we can perfectly recover the full secret support with just one trace per $\alpha \in \mathbb{F}_{2^{12}}$.

Table 6: Quantiles (25th/50th/75th) of Guessing Entropy (log2) for the two Capture Window settings, for $n_a = 1 \dots 5$ attack traces.

Capture Window	n_{attack} 1	2	3	4	5
I	1/1.63/2.32	0.26/0.63/1.03	0.13/0.37/0.72	0/0.20/0.43	0/0.07/0.32
II	0/0.05/0.31	0/0/0.05	0/0/0	0/0/0	0/0/0

5.2 Complexity and performance analysis

Although we have performed our attack on $m = 12$ the complexity analysis we provide here is applicable to other parameter sets such as $m = 13$. We shall decompose the two main steps (profiling and attack) of our attack.

During the profiling step, we compute 2^m parameters (mean vectors and covariance matrices for each α_i). The highest complexity of these parameters is given by the computation of the covariance matrices, which requires $O(n_\mathrm{p} \cdot n_\mathrm{s}^2)$ operations, where n_p is the number of profiling traces used per α and n_s is the number of leakage samples retained for the computation of the covariance. Hence, the profiling step requires $O(2^m \cdot n_\mathrm{p} \cdot n_\mathrm{s}^2)$ operations in total.

For the attack step, LDA reduces the size of the attack traces from n_s to some $\hat{n_\mathrm{s}} \ll n_\mathrm{s}$ (e.g. in some of our experiments from 3839 to 50). LDA also reduces the size of the mean vectors and covariance matrices, which are projected to match the same dimension $\hat{n_\mathrm{s}}$. Furthermore, the efficient linear discriminant from Choudary and Kuhn [CK14] can be precomputed in time $O(\hat{n_\mathrm{s}}^2)$, which is clearly lower than the time required for the computation of the covariance matrices ($O(n_\mathrm{p} \cdot n_\mathrm{s}^2)$) and hence we can include these precomputations as part of the profiling step. We are left with the summation of the n_a attack vectors, which is done in $O(n_\mathrm{a} \cdot \hat{n_\mathrm{s}})$. Hence, the total attack time per α equals $O(2^m \cdot n_\mathrm{a} \cdot \hat{n_\mathrm{s}})$. This implies that the overall attack, for completely retrieving the secret Goppa support is $O(2^{2m} \cdot n_\mathrm{a} \cdot \hat{n_\mathrm{s}})$.

Performance timings: We implemented our attack using Matlab - 9.2.0.538062 (R2017a), with pre-compilation for execution on Intel machines. We have performed timing estimations for both the profiling as well as for the attack phase on a NVIDIA DGX A100 System equipped with two AMD EPYC 7742 CPUs and 2 TB of memory. The profiling step took less than 2000 s in all experiments, while the attack step took on average 50 s per iteration across all values α (≈ 12 ms per α). In terms of computation time, this is roughly equivalent to the time required to recover the entire secret Goppa support $\mathcal{L}$.

Finally, based on our complexity estimates, we can see that going from $m = 12$ to $m = 13$ would only involve a multiplicative factor of 4 in the attack step ($2^{2m} = 2^{24}$ vs $2^{2m} = 2^{26}$), leading to an estimated attack time of roughly $4 \cdot 50$ s ≈ 200 s for $m = 13$.

6 Attack extensions and limitations

Non-valid cipher-text verification We shall first focus on possible countermeasures against our attack. Recall that any plain-text $\boldsymbol{e}$ is encrypted into $\boldsymbol{z} = \boldsymbol{H}_\mathrm{pub}\boldsymbol{e}^T$ which is afterwards taken as input by the Decapsulation algorithm. Hence, all side-channel attacks that fall under the weight one plain-text model, and implicitly our attack, can be avoided using a check function. Indeed, the Decapsulation algorithm could begin with a for loop where we check whether $\boldsymbol{z} = \boldsymbol{H}_\mathrm{pub}[, i]$ for $i \in [\![0; n-1]\!]$. If comparing equality of two mt-length binary vectors can be done in constant time, and since there are n columns, the

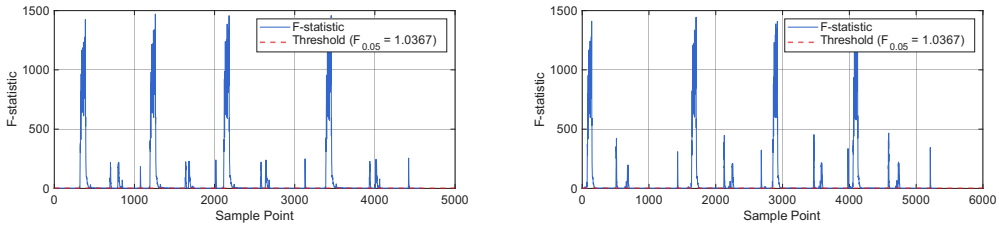


Figure 4: F-test results for Hamming Weight 2, when targeting the product (left) and sum (right) components.

time complexity of this verification is $O(n)$, when $n \to \infty$. Overall, this checkup would not significantly increase the execution time of the Decapsulation.

Since the aforementioned countermeasure can be deployed before the application of the Berlekamp-Massey algorithm, then our attack would fail as well. However, BM leaks information even for higher weights models. Let us detail the case $\mathsf{wt}(\boldsymbol{e}) = 2$.

Template Attack using weight 2 plain-texts Suppose we have a plain-text that satisfies $\mathsf{wt}(\boldsymbol{e}) = 2$, to be more precise, $e_i = e_j = 1$ and $e_l = 0, \forall l \neq i, l \neq j$. This implies that $\boldsymbol{s} = (g(\alpha_i)^{-2}\alpha_i^l + g(\alpha_j)^{-2}\alpha_j^l)_{l=[\![0;2t-1]\!]}$. After the fourth iteration of the Berlekamp-Massey algorithm applied to $\boldsymbol{s}$ we have that $C = [1, \alpha_i + \alpha_j, \alpha_i\alpha_j]$. This implies that for any of the remaining steps the discrepancy will always be equal to 0. Indeed,

$$\begin{aligned} d &= s_{l+1} - s_l \boldsymbol{C_1} - s_{l-1}\boldsymbol{C_2} \\ &= g(\alpha_i)^{-2}\alpha_i^{l+1} + g(\alpha_j)^{-2}\alpha_j^{l+1} - \left(g(\alpha_i)^{-2}\alpha_i^l + g(\alpha_j)^{-2}\alpha_j^l\right)(\boldsymbol{\alpha_i + \alpha_j}) \\ &\quad - \left(g(\alpha_i)^{-2}\alpha_i^{l-1} + g(\alpha_j)^{-2}\alpha_j^{l-1}\right)(\boldsymbol{\alpha_i\alpha_j}) \\ &= 0. \end{aligned}$$

Notice that at each step the values C_1, C_2 are multiplied with different elements in $\mathbb{F}_{2^m}$ (the syndrome components). Such a multiplication is repeated $2t - 4$ times, enough to discriminate the elements $C_1 = \alpha_i + \alpha_j$ and $C_2 = \alpha_i\alpha_j$. In addition, there is a one-to-one mapping between the polynomial $\sigma(x) = x^2 + (\alpha_i + \alpha_j)x + \alpha_i\alpha_j$ and its roots (α_i, α_j).

In practice, one may use our Template Attack strategy to exploit the observed leakage: firstly build profiles for the sum $(\alpha_i + \alpha_j)$ and product $(\alpha_i \cdot \alpha_j)$ separately. Then, for a given trace, find the most likely sums and the most likely products. Afterwards, compute the most probable values for α_i and α_j, in decreasing order of likelihood (trying the most likely combinations).Such approach only requires profiles of the same size as for the experiments for $\mathsf{wt}(\boldsymbol{e}) = 1$ (2^m parameters for the sum and 2^m parameters for the product).

To verify that such attack works indeed, we have used a similar acquisition setup as *Capture Window I* with the oscilloscope configuration C from Table 5. The traces for the sum values included 5300 selected samples, while those for the product values contained 4585. The result of the $F - test$ is shown in Figure 4, while the quartiles of the Guessing Entropy (log2) for the sum $(\alpha_i + \alpha_j)$ and product $(\alpha_i \cdot \alpha_j)$ values are shown in Figure 5. We notice that the median Guessing Entropy is about one when using a single trace (and close to zero when using more than two traces), but using an acquisition setup similar to *Capture Window II* we expect to obtain a median Guessing Entropy of zero even with a single trace for both the sum and product terms.

Extending templates for higher weights We can extend our procedure for even higher Hamming weight in linear time. For example, increasing from weight 2 to weight 3 requires only an extra template. Indeed, for weight 3 plain-texts at the sixth iteration of the BM

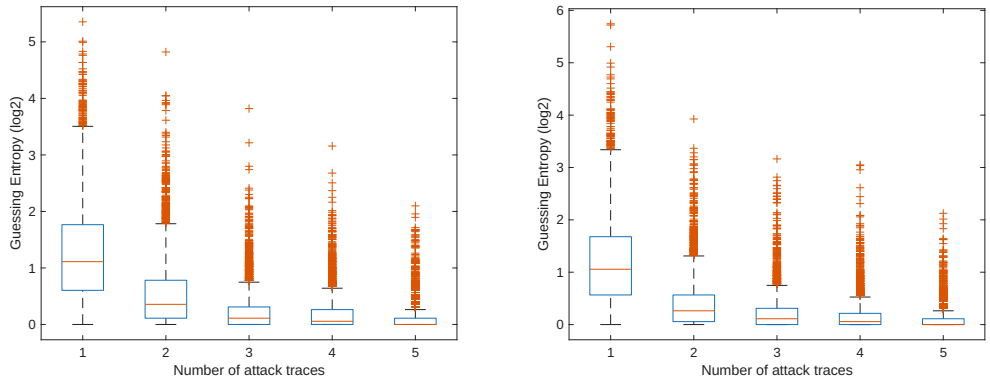


Figure 5: Box plots showing median and 25th/75th percentile of Guessing Entropy for $n_a = 1 \dots 5$ attack traces for all possible values of the product (left) and sum (right).

the connection polynomial will be equal to $C(x) = x^3 + (\alpha_i + \alpha_j + \alpha_k)x^2 + (\alpha_i\alpha_j + \alpha_i\alpha_k + \alpha_j\alpha_k)x + \alpha_i\alpha_j\alpha_k$. Hence, there are only 3 distinct values for which we create our templates. Overall, one needs to compute 3×3^m profiles. The same argument can be extended to even higher weights, implying a linear dependence on 2^m, i.e., $w \times 2^m$, where $w = \mathsf{wt}(\boldsymbol{e})$.

Low weight plain-text check A low weight plain-text verification would induce a significant overhead on the scheme. Indeed, such a procedure verifies whether $\boldsymbol{z} \in \left\{\sum_{i\in\mathcal{I}} \boldsymbol{H}_{\text{pub}}[, i] \mid \mathcal{I} \in \text{subsets}([0, n-1], \leq w)\right\}$ in order to determine if the weight of the plain-text is lower than or equal to w. It should be understood that checking for low-weight cipher-texts becomes prohibitively expensive: enumerating all plain-texts of weight w, which equals $\binom{n}{w}$, already amounts to roughly 6 million checks for $w = 2$ and exceeds 7×10^9 checks for $w = 3$ at the first security level (see Table 7 for $w = 3$.)

Table 7: Number of checks required to verify if the plain-text has weight $1, 2$ or 3.

	$n = 3488$	$n = 6688$	$n = 6960$	$n = 8192$
$\mathsf{wt}(\boldsymbol{e}) = 1$	3488	6688	6960	8192
$\mathsf{wt}(\boldsymbol{e}) = 2$	6081328	22361328	24217320	33550336
$\mathsf{wt}(\boldsymbol{e}) = 3$	7066503136	49835946336	56168037520	91592417280

Theoretical and practical limitations **Side-channel/template limitation** We have seen that in order to build a side-channel attack one requires a minimum number of iterations, say ε, in the BM where some invariants (secret values) are multiplied with syndrome entries. This implies that, theoretically, we could build an attack for plain-text of weight at most $t - \varepsilon$. Indeed, the number of steps where $d = 0$ is, in general, $2t - 2\mathsf{wt}(\boldsymbol{e})$. Hence, for $\mathsf{wt}(\boldsymbol{e}) = t - \varepsilon$ the last $2t - 2(t - \varepsilon) = 2\varepsilon$ iterations are performed with the correct polynomial $C(x)$, e.g., $\mathsf{wt}(\boldsymbol{e}) = 1 \to 2t - 2$ and $\mathsf{wt}(\boldsymbol{e}) = 2 \to 2t - 4$ iterations.

Recall that, for a given $\boldsymbol{e}$ with $\mathsf{wt}(\boldsymbol{e}) = w < t$ with one trace we recover the polynomial $C(x)$. Also, for such a scenario we require $w \times 2^m$ profiles. Next, we compute the roots of this polynomial and obtain the values $\alpha_1, \alpha_2, \dots, \alpha_w$. However, we do not know which value corresponds to which index, i.e., we have to match each value to it's exact position. In order to do so, we will apply an algorithm and determine the number of needed queries to match all n values on their positions.

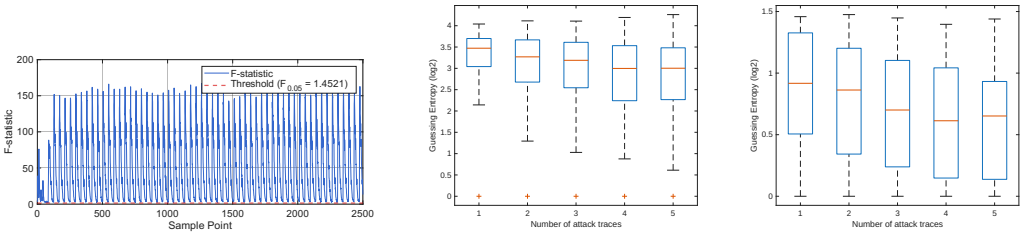


Figure 6: F-test (left) and Template attacks for Cortex-M4 implementation of BM using 32 individual values (middle) and only hamming weights of first 32 values (right).

Number of traces Our algorithm creates a binary matrix which encodes each $\boldsymbol{e}$ ($\mathsf{wt}(\boldsymbol{e}) = w$) needed for the attack. We thus create a binary matrix $\boldsymbol{M}$, by vertically concatenating each vector $\boldsymbol{e}$. Notice that $\boldsymbol{M}$ is a $l \times n$ where l is the number of queries. Each row of $\boldsymbol{M}$ has Hamming weight i and there are a total of $w \times l$ ones. Each column of $\boldsymbol{M}$ has to be unique, since it will denote a signature for its index. Once we built this matrix we recover the position of an element α using the following procedure.
1. Compute $\boldsymbol{M}$ using combinatorial group testing methods [DH00];
2. Apply the attack with $\boldsymbol{e} = \boldsymbol{M}[i,]$ for all $i \in \{1, \ldots, l\}$ which will give l sets $S_1, \ldots S_l$ of private Goppa points;
3. For each $\alpha \in \mathbb{F}_{2^m}$ creates signature $\sigma_\alpha = (b_1, \ldots, b_l)$ where $b_j = 1$ if $\alpha \in S_j$ else $b_j = 0$;
4. Match the value σ_α with a column in $\boldsymbol{M}$, which will give the position of α.

Table 8: Numbers of queries/traces for a full Goppa support recovery

	$n = 3488$	$n = 6688$	$n = 6960$	$n = 8192$
$\mathsf{wt}(\boldsymbol{e}) = 2$	3487	6687	6959	8191
$\mathsf{wt}(\boldsymbol{e}) = t/2$	218	209	234	256
$\mathsf{wt}(\boldsymbol{e}) = t - 20$	159	124	140	152

We need to discriminate n columns hence we require at least $l \geq \log_2(n)$ queries - with l bits we can encode $2^l \geq n$ distinct binary strings. However, such a lower bound cannot be achieved using sublinear/low constant weight rows. If we perform a rough estimation, there are $w^* = w \times l/n$ ones on each column ($w \times l$ is the total number of ones). Therefore, we require $\binom{l}{w^*} \geq n$, i.e., $\binom{l}{\lfloor l \times w/n \rfloor} \geq n$. Notice from Table 8 that with only 218 attack traces one can recover the complete Goppa support, which is significantly smaller than other similar settings [BCM+25, GJJ22b] when plain-text of weight $t/2$ are used. In the asymptotics, if we consider $t = n/\log_2(n)$ and plain-text of weight $t/2$ then we have $l \geq \log_2(n) \log_2^{1.5}(\log_2(n))$.

Attacks on Cortex-M4 implementation of BM To further check if different implementations of BM can be attacked, we have performed a brief evaluation of our Template Attacks against the Cortex-M4 implementation of Chen and Chou [CC21], using $2^5 = 32$ values of α for the profiling and attack. The F-test results are shown in Figure 6 (left). We see the leakage shape is similar to that from the reference implementation and we can still see the peaks for all the multiplication iterations, although the amplitude of the leakage peaks is smaller than for the reference implementation. The template attack results are shown in Figure 6 (middle). This time the attack is less successful, confirming the smaller F-test leakage results, as it only reduces the Guessing Entropy from 5 bits to 3 bits, even when using up to 5 attack traces.

The reduced leakage and attack performance of our standard Template Attack approach

are most likely due to the fact that the BM implementation of Chen and Chou use a fast carry-less multiplication approach, which expands the GF operands into sparse 32-bit representations, allowing the use of the standard multiplication operation from the Cortext M4 processor to compute the carry-less multiplication. This in turn means that for each iteration of the discrepancy computation there are only 4 multiplications performed, out of which only two with the same bits of the secret value α. On the other hand, for the reference implementation, the GF carry-less multiplication is performed in a standard way, by consecutively multiplying and shifting the operands, resulting in 12 multiplications per iteration of the discrepancy computation, hence leading to the higher leakage observed. A possible alternative attack in this case is to target only the Hamming weight values of the secret coefficients, leading to the attack results from Figure 6 (right), where we see that the median Guessing Entropy drops below one even when using a single attack trace, for this smaller 32-group attack. We leave as a scope of further research the possibility of optimizing our Template Attacks for such approach of the GF carry-less multiplication.

Masking Another possible countermeasure would be to apply masking techniques in order to avoid the direct dependency on the secret value α when power consumption is analyzed. As we have shown, the leakage comes from the fact that, when $\mathsf{wt}(\boldsymbol{e})$ is small, few secret values are multiplied with *random* values many times in the BM decoder. Thus, we could try to apply private finite field multiplication techniques [BBP+17]. However, providing a sound result at the end of the BM decoder, when masking is applied at each iteration or globally is not a trivial task.

Another solution would be to use codeword masking [MSS13, TDP20]. This technique was previously mentioned in the context of HQC [GSA+25, MNMD25]. The idea is to add a new codeword (the mask) to the existing cipher before decoding is applied. The design of *Classic McEliece* is clearly not suitable for a direct application of this technique. Indeed, if the receiver computes a mask codeword $\boldsymbol{c} \in \mathscr{G}$ and xor with the cipher-text $\boldsymbol{v}$ then the syndrome $\boldsymbol{s} = \boldsymbol{H}_{\text{priv}} \cdot (\boldsymbol{v} \oplus \boldsymbol{c})^T = \boldsymbol{H}_{\text{priv}}\boldsymbol{v}^T$ since $\boldsymbol{H}_{\text{priv}}\boldsymbol{c}^T = \boldsymbol{0}$. Therefore, our attack still holds under these circumstances.

7 Conclusions

The reference implementation of the *Classic McEliece* KEM was subject to various side-channel attacks, targeting almost all functions in the decapsulation process, except for the support generation and Berlekamp-Massey decoding algorithm. Here, we provide a first side-channel attack targeting the BM algorithm, crafted on the chosen cipher-text model and using small Hamming weight plain-texts.

We have performed efficient template attacks against the reference implementation of the MCELIECE348864 parameter set, running on the ARM Cortex M4 using the Chip Whisperer platform. Using a single attack trace per secret coefficient, we can perfectly recover the entire Goppa secret support.

The vulnerability that we identify in the BM is unavoidable in the current implementation, and no practical means of mitigating it exist as of now without incurring undesirable overhead. Possible paths towards a side-channel resistant implementation include investigation of other decoders, such as the Patterson decoding algorithm or the Interpolation method by Bernstein, as well as implementation variants of the BM.

Acknowledgement The authors were partially supported by RExQTCS: Romanian Excellence Quantum Technologies enhancing Cybersecurity, a grant of the Ministry of Education and Research from Romania, CCCDI - UEFISCDI, project number PN-IV-P6-6.1-CoEx-2024-0214, within PNCDI IV. V-F. Dragoi was partially supported by Aurel Vlaicu University of Arad through the research grant UAV-IRG-1-2025- 12.

A Example on BM computations

Let $m = 3, \mathbb{F}_{2^m} = \mathbb{F}_2/(x^3 + x + 1)$ and $t = 4, g(x) = x^4 + x + 1$. Let $\gamma^3 + \gamma + 1 = 0$ be the generator of $\mathbb{F}_{2^m}$ with $\gamma^3 = \gamma + 1, \gamma^4 = \gamma^2 + \gamma, \gamma^5 = \gamma^2 + \gamma + 1, \gamma^6 = \gamma^2 + 1$. The private parity-check matrix in $\mathbb{F}_{2^m}$, elements in order $\{0, 1, \gamma, \gamma^2, \ldots, \gamma^6\}$, is

$$\boldsymbol{H}_{\text{priv}} = \begin{pmatrix} 1 & 1 & \gamma^2 & \gamma^4 & \gamma^2 & \gamma & \gamma & \gamma^4 \\ 0 & 1 & \gamma^3 & \gamma^6 & \gamma^5 & \gamma^5 & \gamma^6 & \gamma^3 \\ 0 & 1 & \gamma^4 & \gamma & \gamma & \gamma^2 & \gamma^4 & \gamma^2 \\ 0 & 1 & \gamma^5 & \gamma^3 & \gamma^4 & \gamma^6 & \gamma^2 & \gamma \\ 0 & 1 & \gamma^6 & \gamma^5 & 1 & \gamma^3 & 1 & 1 \\ 0 & 1 & 1 & 1 & \gamma^3 & 1 & \gamma^5 & \gamma^6 \\ 0 & 1 & \gamma & \gamma^2 & \gamma^6 & \gamma^4 & \gamma^3 & \gamma^5 \\ 0 & 1 & \gamma^2 & \gamma^4 & \gamma^2 & \gamma & \gamma & \gamma^4 \end{pmatrix}$$

Weight 1 plain-text

Let $\alpha = \gamma^3$, equivalently, the fifth column of $\boldsymbol{H}_{\text{priv}}$. We compute first $g^2(\alpha) = \alpha^8 + \alpha^2 + 1 = \gamma^{24} + \gamma^6 + 1 = \gamma^3 + \gamma^6 + 1 = \gamma + 1 + \gamma^2 + 1 + 1 = \gamma^2 + \gamma + 1 = \gamma^5$. This means $g(\alpha)^{-2} = \gamma^2$. Now computing the syndrome vector corresponding to α we obtain $\boldsymbol{s} = [\gamma^2, \gamma^3 \cdot \gamma^2, \gamma^6 \cdot \gamma^2, \gamma^9 \cdot \gamma^2, \gamma^{12} \cdot \gamma^2, \gamma^{15} \cdot \gamma^2, \gamma^{18} \cdot \gamma^2, \gamma^{21} \cdot \gamma^2] = [\gamma^2, \gamma^5, \gamma, \gamma^4, 1, \gamma^3, \gamma^6, \gamma^2]$. The computations of BM applied to $\boldsymbol{s}$ give the following

- $N = 0$ we get $d = s_0 \cdot 1 = \gamma^2$. The if condition $2L < N$ is satisfied and hence we update the parameters accordingly, $L = 1, B = C, b = \gamma^2, l = 1$. At this step we have $C = [1, \gamma^2, 0, \ldots, 0]$.
- $N = 1$ we get $d = s_1 \cdot 1 + s_0 \cdot C_1 = \gamma^5 + \gamma^2\gamma^2 = \gamma^5 + \gamma^4 = 1$. We update C, at position $l = 1$ with the value $-db^{-1} + C[l]$. In our case we get $C = [1, \gamma^2 - \gamma^5, 0, \ldots, 0] = [1, \gamma^3, 0, \ldots, 0]$. In this case the if condition $2L < N$ is not satisfied and we already have $C = [1, \alpha, 0 \ldots, 0]$.
- $N = 2$ we get $d = s_2 \cdot 1 + s_1 \cdot C_1 = \gamma + \gamma^5\gamma^3 = 0$, and so on until the last step.

Weight 2 plain-text

Take $\alpha_1 = 1, \alpha_2 = \gamma^6$. The syndrome in this case equals $\boldsymbol{s} = [\gamma^5, \gamma, \gamma^6, \gamma^3, 0, \gamma^2, \gamma^4, \gamma^5]$.

- $N = 0$ we get $d = s_0 \cdot 1 = \gamma^5$. We can compute C which equals $C = [1, \gamma^5, 0, \ldots, 0]$. The condition $2L < N$ is satisfied and we update $L = 1, B = C, b = d = \gamma^5, l = 1$
- $N = 1$ we get $d = s_1 \cdot 1 + s_0 \cdot C_1 = \gamma + \gamma^5 \cdot \gamma^5 = \gamma + \gamma^3 = 1$. We update C by $C = [1, \gamma^5 - 1 \cdot \gamma^{-5}, 0, ;0] = [1, \gamma^3, 0, \ldots, 0]$. The condition $2L < N$ is not satisfied, thus we only update $l = 2$.
- $N = 2$ we get $d = s_2 \cdot 1 + s_1 \cdot C_1 = \gamma^6 + \gamma \cdot \gamma^3 = \gamma^3$. We update our C with $C = [1, \gamma^3, -\gamma^3\gamma^{-5}, 0, \ldots, 0] = [1, \gamma^3, \gamma^5, 0, \ldots, 0]$ The condition $2L < N$ is satisfied hence we update $B, b = d = \gamma^3, l = 3$.
- $N = 3$ we get $d = s_3 \cdot 1 + s_2 \cdot C_1 + s_1 \cdot C_2 = \gamma^3 + \gamma^6 \cdot \gamma^3 + \gamma \cdot \gamma^5 = \gamma$. We can update C and obtain $C = [1, \gamma^3 - \gamma \cdot \gamma^{-3}, \gamma^5 - \gamma \cdot \gamma^{-3}\gamma^3, 0, \ldots, 0] = [1, \gamma^2, \gamma^6, 0, \ldots, 0]$.
- $N > 3$ the discrepancy equals zero.

We have obtained $C(x) = x^2 + \alpha^2 x + \alpha^6$ which admits two simple roots $1, \alpha^6$.

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.

[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.

[BBP+17] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 397–426, Cham, 2017. Springer International Publishing.

[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 Transactions on Cryptographic Hardware and Embedded Systems*, 2025(2):94–125, Mar. 2025.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, pages 16–29. Springer, 2004.

[BDK+25] Sebastian Bitzer, Jeroen Delvaux, Elena Kirshanova, Sebastian Maaßen, Alexander May, and Antonia Wachter-Zeh. How to lose some weight: a practical template syndrome decoding attack. *Designs, Codes and Cryptography*, 2025.

[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.

[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+21] Pierre-Louis Cayrel, Brice Colombier, Vlad-Florin Dragoi, Alexandre Menu, and Lilian Bossuet. Message-recovery laser fault injection attack on the Classic McEliece cryptosystem. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic*

Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II, volume 12697 of *Lecture Notes in Computer Science*, pages 438–467. Springer, 2021.

[CDCG22] Brice Colombier, Vlad-Florin Dragoi, Pierre-Louis Cayrel, and Vincent Grosso. Profiled side-channel attack on cryptosystems based on the binary syndrome decoding problem. *IEEE Trans. Inf. Forensics Secur.*, 17:3407–3420, 2022.

[CGCD25] Brice Colombier, Vincent Grosso, Pierre-Louis Cayrel, and Vlad Dragoi. Message-recovery Horizontal Correlation Attack on Classic McEliece. In *Conference on Constructive Approaches for Security Analysis and Design of Embedded Systems (CASCADE 2025)*, Saint-Étienne, France, April 2025.

[Cho15] Marios O. Choudary. Efficient multivariate statistical techniques for extracting secrets from electronic devices. Technical Report UCAM-CL-TR-878, University of Cambridge, Computer Laboratory, September 2015.

[CK14] Marios O Choudary and Markus G Kuhn. Efficient stochastic methods: Profiled attacks beyond 8 bits. In *International Conference on Smart Card Research and Advanced Applications*, pages 85–103. Springer, 2014.

[CK17] Marios O Choudary and Markus G Kuhn. Efficient, portable template attacks. *IEEE Transactions on Information Forensics and Security*, 13(2):490–501, 2017.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[DCCR13] Vlad Dragoi, Pierre-Louis Cayrel, Brice Colombier, and Tania Richmond. Polynomial structures in code-based cryptography. In Goutam Paul and Serge Vaudenay, editors, *Progress in Cryptology – INDOCRYPT 2013*, pages 286–296, Cham, 2013. Springer International Publishing.

[DCV+24] 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 Transactions on Cryptographic Hardware and Embedded Systems*, 2025(1):367–391, Dec. 2024.

[DH00] Ding-Zhu Du and Frank K. Hwang. *Combinatorial Group Testing and Its Applications*, volume 12 of *Series on Applied Mathematics*. World Scientific, Singapore; River Edge, NJ, 2nd edition, 2000.

[DRBL18] Vlad Dragoi, Tania Richmond, Dominic Bucerzan, and Axel Legay. Survey on cryptanalysis of code-based cryptography: From theoretical to physical attacks. In *2018 7th International Conference on Computers Communications and Control (ICCCC)*, pages 215–223, 2018.

[fSidI25] Bundesamt für Sicherheit in der Informationstechnik. Bsi tr-02102-1. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.html, Jan 2025. Standard.

[GCCD23] Vincent Grosso, Pierre-Louis Cayrel, Brice Colombier, and Vlad-Florin Dragoi. Punctured syndrome decoding problem - efficient side-channel attacks against Classic McEliece. In Elif Bilge Kavun and Michael Pehl, editors, *Constructive Side-Channel Analysis and Secure Design - 14th International Workshop, COSADE 2023, Munich, Germany, April 3-4, 2023, Proceedings*, volume 13979 of *Lecture Notes in Computer Science*, pages 170–192. Springer, 2023.

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

[GJJ22a] Qian Guo, Andreas Johansson, and Thomas Johansson. A key-recovery side-channel attack on classic McEliece implementations. *IACR TCHES*, 2022(4):800–827, 2022.

[GJJ22b] Qian Guo, Andreas Johansson, and Thomas Johansson. A key-recovery side-channel attack on Classic McEliece implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):800–827, 2022.

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

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[GSA+25] Guillaume Goy, Maxime Spyropoulos, Nicolas Aragon, Philippe Gaborit, Renaud Pacalet, Fabrice Perion, Laurent Sauvage, and David Vigilant. Side-channel sensitivity analysis on HQC: Towards a fully masked implementation. Cryptology ePrint Archive, Paper 2025/1344, 2025.

[HGDM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011.

[HGG20] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Applications of machine learning techniques in side-channel attacks: a survey. *Journal of Cryptographic Engineering*, 10(2):135–162, 2020.

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

[LNPS20] Norman Lahr, Ruben Niederhagen, Richard Petri, and Simona Samardjiska. Side channel information set decoding using iterative chunking - plaintext recovery from the "Classic McEliece" hardware reference implementation. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 881–910. Springer, 2020.

[LS01] Pierre Loidreau and Nicolas Sendrier. Weak keys in the mceliece public-key cryptosystem. *IEEE Transactions on Information Theory*, 47(3):1207–1211, 2001.

[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.

[MNB+25] Hannah J. Manetsch, Gyohei Nomura, Elie Bataille, Xudong Lv, Kon H. Leung, and Manuel Endres. A tweezer array with 6100 highly coherent atomic qubits. *Nature*, 2025.

[MNMD25] Nathan Maillet, Cyrius Nugier, Vincent Migliore, and Jean-Christophe Deneuville. Key recovery from side-channel power analysis attacks on non-simd hqc decryption. In *Advances in Cryptology – CRYPTO 2025: 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2025, Proceedings, Part V*, page 70–102, Berlin, Heidelberg, 2025. Springer-Verlag.

[MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

[MSS13] Dominik Merli, Frederic Stumpf, and Georg Sigl. Protecting PUF error correction by codeword masking. Cryptology ePrint Archive, Paper 2013/334, 2013.

[oST24a] National Institute of Standards and Technology. Fips 203 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.

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

[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.

[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.

[PPM+23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Computing Surveys*, 55(11):1–35, 2023.

[RPC22] Anca Rădulescu, Pantelimon G Popescu, and Marios O Choudary. Ge vs gm: Efficient side-channel security evaluations on full cryptographic keys. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 886–905, 2022.

[SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 411–425. Springer, 2008.

[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 *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[SMY09] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 443–461. Springer, 2009.

[TDP20] Lars Tebelmann, Jean-Luc Danger, and Michael Pehl. Self-secured puf: Protecting the loop puf by masking. In *Constructive Side-Channel Analysis and Secure Design: 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1–3, 2020, Revised Selected Papers*, page 293–314, Berlin, Heidelberg, 2020. Springer-Verlag.

[VCC+25] Nicolas Vallet, Pierre-Louis Cayrel, Brice Colombier, Vlad-Florin Drăgoi, and Vincent Grosso. Optimizing key recovery in classic mceliece: Advanced error correction for noisy side-channel measurements. *IACR Communications in Cryptology*, 2(2), 2025.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, pages 740–757, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.