IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2022, No. 4, pp. 800–827. DOI:10.46586/tches.v2022.i4.800-827

A Key-Recovery Side-Channel Attack on Classic McEliece Implementations

Qian Guo[1], Andreas Johansson[1] and Thomas Johansson[1]

Dept. of Electrical and Information Technology, Lund University, Lund, Sweden,
{qian.guo,andreas.johansson,thomas.johansson}@eit.lth.se

Abstract. In this paper, we propose the first key-recovery side-channel attack on Classic McEliece, a KEM finalist in the NIST Post-quantum Cryptography Standardization Project. Our novel idea is to design an attack algorithm where we submit special ciphertexts to the decryption oracle that correspond to cases of single errors. Decoding of such ciphertexts involves only a single entry in a large secret permutation, which is part of the secret key. Through an identified leakage in the additive FFT step used to evaluate the error locator polynomial, a single entry of the secret permutation can be determined. Iterating this for other entries leads to full secret key recovery. The attack is described using power analysis both on the FPGA reference implementation and a software implementation running on an ARM Cortex-M4. We use a machine-learning-based classification algorithm to determine the error locator polynomial from a single trace. The attack is fully implemented and evaluated in the Chipwhisperer framework and is successful in practice. For the smallest parameter set, it is using about 300 traces for partial key recovery and less than 800 traces for full key recovery, in the FPGA case. A similar number of traces are required for a successful attack on the ARM software implementation.

Keywords: Code-based cryptography · NIST post-quantum standardization · side-channel attacks · Classic McEliece

1 Introduction

The promise of quantum computing has rapidly changed the focus of research and industry in many areas. The growing interest in applications of quantum computing has led to rapid development of quantum computers in recent years. Experimental quantum computers are developed in the labs of companies such as IBM, Google and Microsoft.

The current solutions for information security are threatened by this progress. In particular, cryptographic primitives that base their security on the difficulty of factoring or the discrete log problem, are no longer secure. Shor's algorithm [Sho94] can be used to break these schemes in polynomial time. Even though a sufficiently large quantum computer may still be many years into the future, information processed today must remain secure also in 10 or 20 years from now. So the development of new security solutions that can withstand the threat of quantum computers is both urgent and of utmost importance.

As a major step in the direction, NIST initiated a few years ago the NIST Post-quantum Cryptography Standardization Project [NIS], here called the NIST PQ project. This is an ongoing evaluation and standardization project for two types of cryptographic primitives, KEMs (Key Encapsulation Mechanisms) and digital signatures. It will eventually set new world standards (technically, US Federal Government standards) for post-quantum secure primitives, in a similar manner as was previously done in the development of AES and SHA-3.

Received: 2022-04-15 Accepted: 2022-06-15 Published: 2022-08-31

Post-quantum secure primitives are most commonly constructed based on either lattice problems or decoding problems in the Hamming metric, referred to as lattice-based crypto or code-based crypto (but there are also e.g. hash-based, multi-variate, isogeny-based). The NIST PQ project is now in its final round (round 3) before standardization and we can find one code-based KEM as finalist and two code-based KEMs as alternate candidates (classified roughly as promising candidates that need more study), BIKE [ABB+20] and HQC [AAB+20].

This paper is about Classic McEliece [ABC+20], which is the code-based KEM finalist, together with three lattice-based KEM finalists, Saber, Kyber, and NTRU [DKR+20, SAB+20, CDH+20]. The Classic McEliece KEM proposal is a modified version of the old McEliece PKC construction from the 70's, using the so-called Niederreiter PKC version and scrambled parity-check matrices from Goppa codes. The security is mainly related to the hardness of decoding random codes as well as distinguishing scrambled Goppa codes from random codes.

Classic McEliece is regarded as a conservative design based on a well-studied problem. It is less efficient compared to lattice-based schemes in implementation and key size but has high confidence in its security. The German Federal Office for Information Security (BSI) in [Ger] suggests to use Classic McEliece [ABC+20] and FrodoKEM [NAB+20] for "long-term confidentiality protection".

While the theoretical security of these post-quantum secure primitives is intensively investigated and small steps forward are continuously taken, the study on the implementation security of these schemes is of equal importance. From a practical perspective, it may even be more important, as information leakage from implementations often lead to actual practical attacks, whereas a successful theoretical attack on a proposed scheme may still be very far from an attack that can actually be done in practice.

Side-channel attacks on implementations of cryptographic primitives, initiated by Kocher [Koc96], contain a plethora of different approaches, such as timing attacks and power attacks, etc. There are also the related fault injection attacks. In a power attack, as used in this paper, the continuous power consumption of the target device with the crypto implementation is measured while the device is executing. The measured power consumption can provide information on secret values in the cryptographic scheme. A successful attack both needs to identify where in the execution to measure, i.e. identifying a useful leakage point, and then to describe an algorithm that uses the received side-channel information and determines secret information in the attacked crypto scheme.

The most powerful and common side-channel attack model is of *profiling* type, meaning that it is assumed that the adversary has access to the target device or some form of a copy of the target device. The adversary can then in an initial profiling step characterize and measure on the device to learn possible dependencies, etc. Whereas so-called template attacks have traditionally been the common approach to profiled attacks, a recently developed and now very common approach is to use machine-learning algorithms. In particular, side-channel attacks based on deep learning have recently gained a lot of attention [MPP16, ZBHV19, KPH+19, NDGJ21, PPM+21]. There are also non-profiled side-channel attacks with deep learning [Tim18, PCBP21].

All the lattice-based KEM finalists in the NIST PQ project have been subjects of side-channel attacks that can recover the secret key [RRCB20, XPRO20, NDGJ21, AR21, REB+22, UXT+22] There are also attacks recovering the secret message [SKL+20]. We now see a race between researchers trying to provide better-protected implementations both in hardware and in software, and researchers trying to find even more sophisticated ways of attacking protected implementations of the finalists [BDK+20, NDGJ21, BDH+21, BGR+21, ABH+22]. However, no key-recovery side-channel attack on Classic McEliece implementations is known, only message recovery attacks. This paper proposes the first key-recovery side-channel attack on Classic McEliece implementations.

1.1 Related works

The first code-based cryptosystem was proposed by McEliece [McE78]. Classic McEliece is a modified version of this original scheme and its latest version is described in [ABC$^+$20]. The official submission of the proposal to round 3 of the NIST PQ project contains also implementations. Other published implementations of Classic McEliece can be found, e.g., FPGA implementations in [WSN17, WSN18] and an ARM Cortex-M4 implementation in [CC21].

Side-channel attacks on McEliece PKC with Goppa codes have previously appeared in [STM$^+$08], which presents an implemented timing attack as well as two other side-channel attacks and related countermeasures: a power attack on the construction of the parity check matrix during key generation and a cache attack on the permutation of code words during decryption. See also improvements in [AHPT11]. Power analysis of an implementation of McEliece PKC on an 8-bit AVR microprocessor was presented in [HMP10].

Side-channel attacks on Classic McEliece have previously appeared as a message-recovery attack using a type of reaction attack in [LNPS20]. There has also been a message-recovery laser fault-injection attack described in [CCD$^+$21]. In [LNPS20] the attack is based on [SSMS10] but adapting it to an EM side-channel on a constant-time Berlekamp-Massey decoder. The attack targets the FPGA implementation of Classic McEliece as in [WSN18]. A new message-recovery attack [CDCG22] targeting the encryption process of Classic McEliece was proposed very recently.

Key-recovery reaction attacks have appeared on code-based primitives, most notably on QC-MDPC [GJS16, GJW19] and on QC-LDPC [FHS$^+$17]. These attacks have a close connection to side-channel attacks and have appeared on the NIST PQ project candidate HQC [GJ20] and key-recovery timing attacks have appeared on BIKE and HQC [GHJ$^+$21].

Relevant general timing attacks on schemes using the FO transformation appeared in [GJN20]. Ueno et al. concluded in [UXT$^+$22] that all round-3 NIST PQC candidates but Classic McEliece are vulnerable to implementation attacks by focusing on the FO transformation.

1.2 Contributions

In this paper, we propose the first key-recovery side-channel attacks on Classic McEliece implementations. This is based on an identified general vulnerability caused by the current algorithm design, that can be explored in a side-channel attack. The vulnerability comes from the fact that if the error locator polynomial is fixed, then the additive FFT evaluation procedure in the decoding step behaves the same. We list some main contributions of the paper as follows.

- We present the first key-recovery side-channel attacks on implementations of Classic McEliece both in hardware (FPGA) and in software (ARM Cortex-M4)
- We highlight an identified side-channel vulnerability in the constant-time (Goppa) decoding step that involves an FFT computation for the evaluation of the error locator polynomial. This has to be addressed when designing a protected implementation of Classic McEliece or similar schemes.
- We show the design of a detailed attack algorithm that finds ways of minimizing the number of required traces.

We have applied this attack to the FPGA reference implementation[1] of Classic McEliece and fully implemented and evaluated the different steps. We also apply it to a third-

[1] This hardware implementation is referenced in the official document [ABC$^+$20] of round-3 Classic McEliece and is named “reference implementation” in [LNPS20].

party implementation for the ARM Cortex-M4 CPU [CC21] with full implementation and evaluation.

1.2.1 New techniques

The main idea of the attack is that if the error locator polynomial is fixed, then the later step of an additive FFT to evaluate the error locator polynomial over all the 2^m points ($m = 12$ or 13) is a fixed process. In the FPGA implementation, it corresponds to 1095 clock cycles of computation. If we generate error vectors with only one position in error, in position i, then there exit only 2^m possible error locator polynomials since it is a polynomial of the form $x - \alpha_i$, where α_i is an unknown value. We use a machine-learning-based classification algorithm to determine the error locator polynomial from power measurements. The error locator polynomial outputted by the Berlekamp-Massey algorithm is given by a selected error location after the secret mapping, related to the secret α_i values. We can thus recover parts of the secret support by repeatedly submitting ciphertexts with a single error in different positions and after a few hundred such submissions we can successfully recover the entire secret Goppa polynomial. To do full secret key recovery, we need additional traces.

1.3 Organization

The remaining of the paper is organized as follows. In Section 2, we give the necessary background in coding theory and code-based cryptography. We then describe the novel ideas in Section 3 and present the new attack in detail in Section 4. This is followed by Section 5 showing the experimental results. We conclude the paper and discuss possible improvements and future works in Section 6.

2 Background

In this section, we briefly introduce background information including basics in coding theory, code-based cryptography, and side-channel analysis.

2.1 Notations

We adopt some of the notations in the design document of round-3 Classic McEliece [ABC+20]. We employ bold-face capital characters for matrices and bold-face low-case characters for vectors throughout the paper. Let q be a prime or a power of prime. We denote $\mathbb{F}_q$ the finite field of order q and $\mathbb{F}_q[x]$ the polynomial ring over $\mathbb{F}_q$. The notation $\#\{\mathcal{A}\}$ means the number of elements in the set $\mathcal{A}$. The Hamming weight of a vector $\mathbf{v}$ (denoted $w_{\mathrm{H}}(\mathbf{v})$) is defined as the number of non-zero coordinates of $\mathbf{v}$. The Hamming distance of two vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ (denoted $d_{\mathrm{H}}(\mathbf{v}_1, \mathbf{v}_2)$) is defined to be the number of coordinates in which $\mathbf{v}_1$ and $\mathbf{v}_2$ differ. We use $|x|$ to denote the absolute value of x.

2.2 Coding theory

Linear codes Let $\mathcal{C}$ be a subspace of $\mathbb{F}_q^n$ with dimension k. Then $\mathcal{C}$ is called an $[n, k]_q$ linear code of length n and dimension k. The redundancy of $\mathcal{C}$ is then $r = n - k$. We call a vector $\mathbf{c} = (c_1, \ldots, c_n) \in \mathcal{C}$ a codeword of $\mathcal{C}$, and the support of a codeword $\mathbf{c}$ is defined as the index set $\mathcal{I}(\mathbf{c})$ that

$$\mathcal{I}(\mathbf{c}) = \{i : i \in \{1, \ldots, n\} \text{ and } c_i \neq 0\}.$$

Thus, we have $w_{\mathrm{H}}(\mathbf{c}) = \#\{\mathcal{I}(\mathbf{c})\}$. The minimum distance of a linear code $\mathcal{C}$ is defined as the smallest Hamming distance between two distinct codewords. Let $\mathbf{G}$ be a $k \times n$ matrix

Table 1: Classic McEliece parameter sets.

	m	t	n	$k = n - mt$	level
kem/mceliece348864	12	64	3488	2720	1
kem/mceliece460896	13	96	4608	3360	3
kem/mceliece6688128	13	128	6688	5024	5
kem/mceliece6960119	13	119	6960	5413	5
kem/mceliece8192128	13	128	8192	6528	5

over $\mathbb{F}_q$ whose rows are the vectors of a basis of $\mathcal{C}$. We call $\mathbf{G}$ a generator matrix and the linear code $\mathcal{C}$ is generated by

$$\mathcal{C} = \{\mathbf{uG} : \mathbf{u} \in \mathbb{F}_q^k\}.$$

We can also define $\mathcal{C}$ by an $r \times n$ matrix $\mathbf{H}$, called parity-check matrix, as

$$\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_q^n : \mathbf{Hc}^{\mathrm{T}} = \mathbf{0}\},$$

i.e. $\mathcal{C}$ is the kernel of $\mathbf{H}$. The syndrome of a vector $\mathbf{v} \in \mathbb{F}_q^n$ is defined as $\mathbf{Hv}^{\mathrm{T}}$.

Binary Goppa codes Classic McEliece employs irreducible binary Goppa codes defined as follows.

Definition 1 (Binary Goppa Codes)**.** The binary Goppa code $\mathcal{C}$ over $\mathbb{F}_{2^m}$ is defined by a support vector $\mathbf{p} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_{2^m}^n$, where $\alpha_i \neq \alpha_j$ for $i \neq j$ and the Goppa polynomial $g(x) \in \mathbb{F}_{2^m}[x]$ with degree t. The code $\mathcal{C}$ is the set of codewords $\mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{F}_2^n$ such that

$$\sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} \equiv 0 \mod g(x). \tag{1}$$

We say that the code $\mathcal{C}$ is defined by $\Gamma = (g, (\alpha_1, \ldots, \alpha_n))$. If the Goppa polynomial $g(x)$ is irreducible, then the Goppa code $\mathcal{C}$ has minimum distance $2t + 1$ and is called an irreducible binary Goppa code.

For more information on Goppa codes and their related decoding algorithms, we refer to any textbook on the subject, like [MS78].

2.3 Classic McEliece

The first code-based cryptosystem was proposed by McEliece in 1978 [McE78] using a randomly chosen irreducible binary Goppa code. Later in 1986, Niederreiter [Nie86] proposed a dual variant of the McEliece cryptosystem that uses a parity-check matrix for encryption (rather than using a generator matrix). His original version employing Reed-Solomon codes was attacked in [SS92], but the version with irreducible binary Goppa codes is still secure. Also, it was proven in [LDW94] that McEliece and Niederreiter cryptosystems are equivalent.

Classic McEliece, one of the three KEM/PKE finalists in the NIST PQ project, is built upon the Niederreiter framework. The proposal [ABC+20] provides an IND-CCA2-secure KEM called the Classic McEliece KEM, which is obtained after a standard transformation of an IND-CPA-secure Niederreiter-style PKE (Public-Key Encryption scheme). The parameters of Classic McEliece[2] are shown in Table 1, where m determines the size of the

[2] The Classic McEliece KEM also designed another type of parameter sets marked by “-f”, e.g. kem/mceliece8192128f. The difference is in the key generation procedure. The new attack also applies to the parameter sets in the “-f” class, since there is no difference in the encryption and decryption when the key pair is generated.

binary field, t represents the number of correctable errors, and n the length of the code. Next, we describe the IND-CPA-secure PKE.

Key generation First choose a random irreducible polynomial $g(x) \in \mathbb{F}_{2^m}[x]$ of degree t and a list of distinct elements $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_{2^m}^n$. Thus, we have picked a random binary irreducible binary Goppa code, which serves as the private key of the PKE. We then compute a $t \times n$ parity check matrix $\mathbf{H}_{\mathsf{goppa}}$ over $\mathbb{F}_{2^m}$ and transform it to a $tm \times n$ binary matrix $\mathbf{H}'_{\mathsf{goppa}}$ via replacing each entry in $\mathbf{H}_{\mathsf{goppa}}$ by an m-bit column over $\mathbb{F}_2$. We write the matrix $\mathbf{H}'_{\mathsf{goppa}}$ in the systematic form $\mathbf{H}''_{\mathsf{goppa}} = [\mathbf{I}_{mt}|\mathbf{T}_{mt\times(n-mt)}]$ and set the public key to be $\mathbf{T}$. This step of systematizing $\mathbf{H}'_{\mathsf{goppa}}$ reduces the public key size since it is unnecessary to store or communicate the identity matrix.

The private key of the Classic McEliece KEM contains an additional uniform random n-bit string, which is only used in the CCA transform in case the decapsulation fails.

Algorithm 1 Ken Generation for the PKE

```
Input: The Classic McEliece parameters: $m$, $t$, and $n$
Output: The secret key $(g(x), (\alpha_1, \alpha_2, \ldots, \alpha_n))$ and public key $\mathbf{T}$
Randomly choose a list of distinct elements $(\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_{2^m}^n$ as support
Choose a random irreducible polynomial $g(x) \in \mathbb{F}_{2^m}[x]$ of degree $t$
Compute the $t \times n$ parity-check matrix
    $\mathbf{H}_{\mathsf{goppa}} = \begin{bmatrix} 1/g(\alpha_1) & 1/g(\alpha_2) & \cdots & 1/g(\alpha_n) \\ \alpha_1/g(\alpha_1) & \alpha_2/g(\alpha_2) & \cdots & \alpha_n/g(\alpha_n) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1}/g(\alpha_1) & \alpha_2^{t-1}/g(\alpha_2) & \cdots & \alpha_n^{t-1}/g(\alpha_n) \end{bmatrix}$
Transform $\mathbf{H}_{\mathsf{goppa}}$ to a $tm \times n$ binary matrix $\mathbf{H}'_{\mathsf{goppa}}$ via replacing each entry in $\mathbf{H}_{\mathsf{goppa}}$ by an $m$-bit column over $\mathbb{F}_2$
Transform $\mathbf{H}'_{\mathsf{goppa}}$ in the systematic form $\mathbf{H}''_{\mathsf{goppa}} = [\mathbf{I}|\mathbf{T}]$
Return the secret key $(g(x), (\alpha_1, \alpha_2, \ldots, \alpha_n))$ and public key $\mathbf{T}$
```

Encryption From the public key $\mathbf{T}$, the sender can re-construct the Goppa parity check matrix $\mathbf{H}''_{\mathsf{goppa}}$ in the systematic form $[\mathbf{I}_{mt}|\mathbf{T}_{mt\times(n-mt)}]$. The sender then chooses an error vector $\mathbf{e} \in \mathbb{F}_2^n$ with $w_{\mathrm{H}}(\mathbf{e}) = t$ as the plaintext and produces the syndrome $\mathbf{s} = [\mathbf{I}|\mathbf{T}]\mathbf{e}^{\mathrm{T}}$ as the ciphertext.

Algorithm 2 Encryption for the PKE

```
Input: Plaintext $\mathbf{e} \in \mathbb{F}_2^n$ with $w_{\mathrm{H}}(\mathbf{e}) = t$ and the public key $\mathbf{T}$
Output: Ciphertext $\mathbf{s}$
Return $\mathbf{s} = [\mathbf{I}|\mathbf{T}]\mathbf{e}^{\mathrm{T}}$
```

Decryption The decryption is equivalent to the syndrome decoding of binary Goppa codes, including the steps of computing the syndrome polynomial and the error locator polynomial and that of evaluating the error locator polynomial at the points in $\mathbb{F}_{2^m}$. As in [BCS13, WSN18], official implementations of Classic McEliece employ a decoder from the constant-time Berlekamp-Massey (BM) algorithm. Thanks to a trick attributed to Sendrier in [HG13], one could correct t errors by computing a double-size $2t \times n$ parity-check matrix $\mathbf{H}^{(2)}_{\mathsf{goppa}}$ over $\mathbb{F}_{2^m}$. We transform $\mathbf{H}^{(2)}_{\mathsf{goppa}}$ to a $2mt \times n$ binary parity check matrix $\mathbf{H}'^{(2)}_{\mathsf{goppa}}$

by replacing each entry with a column of m bits. The double-size syndrome $\mathbf{s}^{(2)}$ is then computed as $\mathbf{H}'^{(2)}_{\mathsf{goppa}}[\mathbf{s}|\mathbf{0}]^{\mathrm{T}}$, where we append $n - mt$ zeros to the syndrome $\mathbf{s}$. We then use the constant-time BM algorithm to compute the error locator polynomial $\sigma(x) \in \mathbb{F}_{2^m}[x]$ of $\mathbf{s}^{(2)}$ and evaluate $\sigma(x)$ in all elements in $\mathbb{F}_{2^m}$. This polynomial evaluation over the whole finite field $\mathbb{F}_{2^m}$ can be efficiently implemented through the additive FFT (Fast Fourier Transform) procedure. In the last step, we read the partial secret key $(\alpha_1, \ldots, \alpha_n)$ and check whether $\sigma(\alpha_i) = 0$. We set the i^{th} bit $\mathbf{e}_i = 1$ if $\sigma(\alpha_i) = 0$ and $\mathbf{e}_i = 0$ otherwise.

Algorithm 3 Decryption for the PKE

```
Input:  Ciphertext $\mathbf{s}$ and the secret key $(g(x), (\alpha_1, \alpha_2, \ldots, \alpha_n))$
Output: Plaintext $\mathbf{e}$
1: Compute a double-size $2t \times n$ parity-check matrix
   $$\mathbf{H}^{(2)}_{\mathsf{goppa}} = \begin{bmatrix} 1/g^2(\alpha_1) & 1/g^2(\alpha_2) & \cdots & 1/g^2(\alpha_n) \\ \alpha_1/g^2(\alpha_1) & \alpha_2/g^2(\alpha_2) & \cdots & \alpha_n/g^2(\alpha_n) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{2t-1}/g^2(\alpha_1) & \alpha_2^{2t-1}/g^2(\alpha_2) & \cdots & \alpha_n^{2t-1}/g^2(\alpha_n) \end{bmatrix}$$
2: Transform $\mathbf{H}^{(2)}_{\mathsf{goppa}}$ to a $2mt \times n$ binary parity check matrix $\mathbf{H}'^{(2)}_{\mathsf{goppa}}$ by replacing
   each entry with a column of m bits
3: Compute the double-size syndrome $\mathbf{s}^{(2)} = \mathbf{H}'^{(2)}_{\mathsf{goppa}}[\mathbf{s}|\mathbf{0}]^{\mathrm{T}}$
4: Use the BM algorithm to compute the error locator polynomial $\sigma(x)$
5: Evaluate the polynomial $\sigma(x)$ at $(\alpha_1, \ldots, \alpha_n)$ and recover the plaintext $\mathbf{e}$
6: Return the plaintext $\mathbf{e}$
```

Berlekamp-Massey algorithm and the additive FFT Given a double-size syndrome vector $\mathbf{s}^{(2)}$, Berlekamp-Massey algorithm [Mas69] is employed for computing the error locator polynomial whose roots are the error locations. The error correction capability is t since the size of the double-size syndrome vector $\mathbf{s}^{(2)}$ is $2t$. The BM algorithm can be made constant-time due to its simplicity. We compute the syndrome polynomial from $\mathbf{s}^{(2)}$, which is the input to the BM algorithm. The algorithm initializes polynomials $\sigma(x) = 1 \in \mathbb{F}_{2^m}[x]$, $\beta(x) = x \in \mathbb{F}_{2^m}[x]$, integers $l = 0$ and $\delta = 1 \in \mathbb{F}_{2^m}$ and updates the 4-tuple $(\sigma(x), \beta(x), l, \delta)$ during the k^{th} iteration for $0 \leq k \leq 2t - 1$, according to certain updating rules. The final output, i.e., the found error locator polynomial, is the updated polynomial $\sigma(x)$ after the $2t$ iterations.

Another important problem is to evaluate a polynomial at multiple points, which is solved by the additive FFT algorithm in the Classic McEliece. We focus on the decryption algorithm, in which one needs to evaluate the error locator polynomial at the secret support $(\alpha_1, \ldots, \alpha_n)$. The additive FFT procedure includes two steps, the radix conversion and twisting step transferring the input polynomial $\sigma(x)$ to many 1-coefficient constant polynomials and the reduction step iteratively evaluating at the input points using these constant polynomials.

2.4 Relations between secret key parts in Classic McEliece

If you know the public key $\mathbf{T}$ and some part of the secret key $(g(x), (\alpha_1, \alpha_2, \ldots, \alpha_n))$, can you then efficiently determine other parts of the secret key? Some brief facts are described.

The support splitting algorithm The support splitting algorithm [Sen00] proposed by Sendrier is designed to solve the code equivalence problem of determining if a linear code $\mathcal{C}_1$ can be obtained by the index permutation of another linear code $\mathcal{C}_2$. The input to the support splitting algorithm is two generator matrices and the output is the found permutation. For random linear codes, the dominant cost of the support splitting algorithm is $\mathcal{O}\left(n^3\right)$ with overwhelming probability, where n is the length of the code.

Key recovery The key recovery problem of Classic McEliece is the recovery of the Goppa polynomial $g(x)$ and the vector $\mathbf{p} = (\alpha_1, \ldots, \alpha_n)$ since such information is sufficient for decrypting ciphertexts. The key recovery problem has been investigated in [Sen00, LS01, OS09]. We can determine the polynomial $g(x)$ from the vector $\mathbf{p}$ or determine the vector $\mathbf{p}$ from $g(x)$ and the set $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. If $n = 2^m$, the set $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ is the whole finite field $\mathbb{F}_{2^m}$, and thus, it is sufficient to recover $g(x)$. The whole secret key $(g(x), \mathbf{p} = (\alpha_1, \ldots, \alpha_n))$ can then be recovered by the support splitting algorithm. We just construct a generator matrix $\mathbf{G}_0$ of the Goppa codes from $g(x)$ and an arbitrary support $\mathbf{p}_0$ over the set $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$. Feeding $\mathbf{G}_0$ and a generator matrix $\mathbf{G}_{\mathsf{goppa}}$ from $\mathbf{H}''_{\mathsf{goppa}}$ to the support splitting algorithm, we could reconstruct the secret support.

The Classic McEliece submission proposed four parameter sets with $n < 2^m$, which can provide additional security against key recovery attacks since it is non-trivial to recover the set $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ from $g(x)$ if $2^m - n$ is not small. We return to this problem in Section 4.2.

2.5 Information set decoding (ISD)

A fundamental problem in code-based cryptography is the *syndrome decoding problem*, where one needs to find an unknown $\mathbf{e_0}$ with $w_{\mathrm{H}}(\mathbf{e_0}) = w$, assuming that a parity-check matrix $\mathbf{H}$ and a syndrome $\mathbf{s} = \mathbf{He_0}^{\mathrm{T}}$ are given.

Prange [Pra62] initiated the research line called information set decoding, where the basic idea is to find k error-free coordinates that carry sufficient information to recover the full error vector $\mathbf{e}$. These k coordinates are called the *information set*. This algorithm was further improved by a number of algorithms (e.g., [LB88, Ste88, MMT11, BJMM12, MO15]). Since one part of the new attack is based on an ISD algorithm and we instantiate it with Stern's algorithm [Ste88] for simplicity, we involve a detailed description and complexity analysis of Stern's variant as follows.

Stern's algorithm [Ste88] Stern firstly introduced the idea of using the birthday paradox in information set decoding. We start with a permutation to write the parity check matrix $\mathbf{H}$ in a systematic form $(\mathbf{H_0}, \mathbf{I})$, so the first k coordinates form an information set. We denote $\hat{\mathbf{H}}_0$ (or $\hat{\mathbf{s}}$) the first l columns of $\mathbf{H_0}$ (or rows of $\mathbf{s}$). We then enumerate $\mathbf{e}$ of dimension $k/2$ and weight p, compute $\hat{\mathbf{H}}_0(\mathbf{e}, \mathbf{0})^{\mathrm{T}}$ and $\hat{\mathbf{s}} - \hat{\mathbf{H}}_0(\mathbf{0}, \mathbf{e})^{\mathrm{T}}$, and search for collisions. Last, we check whether the weight of remaining $(r - l)$ coordinates of the obtained error vector is $(w - 2p)$.

The list size is $\binom{k/2}{p}$. The complexity of one iteration of Stern's algorithm is

$$W = C_{\mathsf{Gauss}} + 2(n-k) \cdot \binom{k/2}{p} + (n-k-l) \cdot \binom{k/2}{p}^2 2^{-l},$$

where C_{Gauss} is the cost of Gauss Elimination that can be set as $0.5 \cdot (n-k)k^2$, if we use a basic school book form of the algorithm.

The complexity of Stern's algorithm can be written as W/P, where P is the probability that we find one solution in one iteration. Since in our problem setting the weight w is larger than a threshold called *GV-bound*, there exist many solutions. Then, the probability P can be estimated as

$$P \approx \binom{k/2}{p}^2 \binom{n-k-l}{w-2p} 2^{-(n-k)}.$$

2.6 Neural-network-aided profiled side-channel analysis

A profiled side-channel attack consists of a profiling stage and an attack stage. Typically, a device ideally identical to the intended target is used during profiling where the attacker has full control and can set the inputs to the device, like the secret key and the ciphertext. A large number of traces are captured through side-channel leakage while the device performs a cryptographic operation with inputs picked by the attacker. Each trace is labeled with a piece of information that is related to the selected input. The set of traces and labels are then used to construct a model, that based on an observed trace estimates the true label of the trace. At the attack stage, the model is used to classify traces captured from the device under attack that could be the same or different from the profiling device.

For profiling, templates introduced by [CRR03] have been used to model the relation between observed traces and labels. For this type of profiling, the estimated noise of a trace is used to determine the most probable label. Machine learning techniques, such as support vector machines and random forest has been used for profiling to get around some of the shortcomings of the template techniques [LPB+15]. With the rapid development of deep learning, neural networks have shown promising results for profiled side-channel attacks [KPH+19].

Common architectures for neural networks in the context of side-channel attacks are the convolutional neural network (CNN) and the multilayer perceptron (MLP). CNN's have shown to be less sensitive to jitter, i.e., when traces are misaligned due to clock phase variation or intentional phase variations introduced as countermeasures [CDP17]. In the case of well-synchronized traces, the MLP has shown to be effective for profiling [RJJ+19, Mag19].

An MLP consists of several layers where the first layer is called the input layer and the last is called the output layer. Layers in between, are called hidden layers. Rather simple MLP's i.e., shallow neural networks with only a few hidden layers, have successfully been used to conduct side-channel attacks [NDGJ21]. Each layer in an MLP consists of several neurons that are fully connected to neurons in the previous layer. During a supervised training of an MLP, traces are fed to the input layer and the predicted labels at the output layer are compared to the true labels. An optimizer is then used to tune the connection between the neurons such that the MLP becomes more accurate in predicting labels of traces.

2.7 Test vector leakage assessment

During the NIST *Non-Invasive Attack Testing Workshop* in 2011, Goodwill et al. [GJJR11] presented the Test Vector Leakage Assessment (TVLA) as a metric to evaluate side-channel leakage. The TVLA has been used to evaluate implemented side-channel countermeasures [BGN+14, BGG+15], and to locate points of interest during attacks [RJJ+19].

During a TVLA, side-channel measurements are divided into two sets and Welch's t-test is applied at each sampling point to determine if the two sets are different by evaluating the null hypothesis that the two sets have equal mean. To perform the t-test on the two sets of measurements $\mathcal{T}_0$ and $\mathcal{T}_1$, the test statistic t_{obs} of Welch's t-test is calculated as

$$t_{obs} = \frac{\hat{\mu}_0 - \hat{\mu}_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}}$$

where $\hat{\mu}_i$, s_i, and n_i are the mean, standard deviation, and cardinality of $\mathcal{T}_i$. If $|t_{obs}| \geq 4.5$, the null hypothesis is rejected at a confidence level of 99.998% if $s_0 \approx s_1$, and $n_0 \approx n_1 \geq 100$. In the context of side-channel analysis, a rejected null hypothesis suggests that the two sets of measurements are noticeably different and leak side-channel information that possibly could be exploited.

3 A basic description of the new attack idea

In this section, we briefly describe the new attacking idea. We start with the attack model and then give the essence of the new key-recovery attack.

3.1 The threat model

The Classic McEliece KEM is designed for IND-CCA2 security. In this paper, we further study its side-channel resilience when being implemented in low-end devices and assume that the adversary is capable of measuring the power traces during the decapsulation process. The adversary aims to recover the secret key via:

1. the adversary firstly selects ciphertexts satisfying certain properties and sends these ciphertexts to the Classic McEliece KEM decapsulation device;

2. the adversary could physically observe the power traces.

Furthermore, the adversary is assumed to have a similar device/environment running the Classic McEliece KEM, and thus the adversary could perform profiling activities. Note that we do NOT assume the same secret key is cloned to the profiling environment. The difficulty of the attack is to design ciphertext properties for profiling to facilitate the key recovery via side channels.

3.2 The essence of the attack

The secret key of the classic McEliece KEM consists of an irreducible polynomial $g(x)$, a vector $\mathbf{p} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ where $\alpha_i \in \mathbb{F}_{2^m}$ and $\alpha_i \neq \alpha_j$ for $i \neq j$, and one uniform random n-bit string. As the random string is only used when the KEM fails, the key recovery problem is to recover the polynomial $g(x)$ and the secret support $\mathbf{p} = (\alpha_1, \ldots, \alpha_n)$. We aim to first partially recover the secret support $\mathbf{p} = (\alpha_1, \ldots, \alpha_n)$ and list the main observation below.

Observation 1. *A long part of the computation in the decapsulation algorithm behaves the same if the error locator polynomial is fixed. For instance, the additive FFT procedure to evaluate the error locator polynomial $\sigma(x)$ over the whole finite field only depends on $\sigma(x)$.*

This fact generally holds for common implementations of Classic McEliece decryption, including the known FPGA and ARM Cortex-M4 implementations considered in this paper. We show as an example the leakage in a recent Cortex-M4 implementation published at CHES 2021 [CC21], given in Listing 1.

```
1 int decrypt_n3488_t64(unsigned char *e, const unsigned char *sk, const
    unsigned char *s){
2   ...
3   bm( locator , s_priv );    // find error locator
4   fft_p64_v4096_u32( temp , locator );   // find the root of error locator
5
6   // need to know the position only
7   for(i=0;i<128;i++) error[i] = get_nonzero_mask_u32( temp[i] , GFBITS ) ^ 0
      xffffffff;
8   ...
9 }
```

Listing 1: Part of the decryption function in [CC21]

In Line 3, the Berlekamp-Massey (BM) algorithm computes the error locator polynomial $\sigma(x)$ stored in a variable called "locator". Then, the computations in Line 4 and Line 7 only depend on the value stored in the "locator", i.e., $\sigma(x)$.

To evaluate $\sigma(x)$ at all elements of $\mathbb{F}_{2^m}$, both the FPGA and ARM Cortex-M4 implementations considered in this paper make use of an additive FFT to speed up the computation. The additive FFT algorithm takes as input, the polynomial $\sigma(x)$ of degree at most t, and outputs the value of $\sigma(\beta)$ for all $\beta \in \mathbb{F}_{2^m}$. The main idea of the additive FFT, is to exploit that $(\beta+1)^2 + (\beta+1) = \beta^2 + \beta$ for $\beta \in \mathbb{F}_{2^m}$. Thus, by doing a radix conversion and writing $\sigma(x) = \sigma^{(0)}(x^2+x) + x\sigma^{(1)}(x^2+x)$ where $\deg(\sigma^{(0)}) = \lfloor \deg(\sigma)/2 \rfloor$ and $\deg(\sigma^{(1)}) = \lfloor (\deg(\sigma)-1)/2 \rfloor$, the value of $\sigma(\beta+1)$ can be easily calculated by first calculating $\sigma(\beta) = \sigma^{(0)}(\beta^2+\beta) + \beta\sigma^{(1)}(\beta^2+\beta)$, and then using $\sigma(\beta+1) = \sigma(\beta) + \sigma^{(1)}(\beta^2+\beta)$. Thus, $\sigma(x)$ only needs to be evaluated at half of the elements of $\mathbb{F}_{2^m}$, since the other half, which contains "1", can easily be calculated from the first half. By twisting the basis of $\sigma^{(0)}(x)$ and $\sigma^{(1)}(x)$, half of the elements where the polynomials should be evaluated will contain the "... + 1". Thus, the radix conversion can performed again to get $\sigma^{(0)}(x) = \sigma^{(0,0)}(x^2+x) + x\sigma^{(0,1)}(x^2+x)$ and $\sigma^{(1)}(x) = \sigma^{(1,0)}(x^2+x) + x\sigma^{(1,1)}(x^2+x)$.

The additive FFT repeats the twisting and radix conversion recursively until we are left with constant polynomials. These constants are then read and we recursively evaluate $\sigma(x)$. For our attack, the important point is that the value of the constants, as well as the calculation of all intermediate polynomials, only depends on the $\sigma(x)$ that we feed to the additive FFT.

New idea for a profiled attack Based on Observation 1, we create ciphertexts from plaintexts $\mathbf{e}$'s with $w_{\mathrm{H}}(\mathbf{e}) = 1$; for the decryption of such a ciphertext, the computed error locator polynomial $\sigma(x)$ is a monic polynomial of degree 1 and up to q error locator polynomials are possible. We then could design a profiled attack to recover $\sigma(x)$. The basic idea of the attack is described in two phases as follows.

Profiling phase: We randomly sample secret supports $\mathbf{p}^{\mathsf{pub}}$. We then sample error vectors $\mathbf{e}_i$ such that all the entries are zero except for the i^{th} one. Then, the non-zero position will lead to an error locator polynomial $\sigma(x) = (x - \mathbf{p}^{\mathsf{pub}}(i))$. Thus, we have $q = 2^m$ different error locator polynomials and could allocate all the traces in q categories according to the corresponding error locator polynomial. We then train a neural network to classify the traces from the q different categories, that are expected to have distinctive traces as explained later.

Attacking phase: In each decryption oracle call, we send an error vector $\mathbf{e}_i$. The error locator polynomial can be computed as $\sigma(x) = x - \alpha_i$. By the classification model built in the profiling phase, we could detect $\sigma(x)$ and therefore recover α_i. After trying all the possible i's, we could recover the secret support $\mathbf{p}$, so the required number of traces in the attack phase is at least n. However, not all of the α_i's in the secret support are required to re-build the irreducible polynomial $g(x)$. With this observation, we could reduce the required number of test traces.

Recover $g(x)$ by polynomial factorization Next, we show how to determine the irreducible polynomial $g(x)$, once the partial secret-key $\mathbf{p}$ is recovered. We use one valid ciphertext $\mathbf{c} = (c_1, c_2, \ldots, c_n)$ and compute

$$c(x) = \sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} \prod_{j \in \mathcal{I}(\mathbf{c})} (x - \alpha_j), \tag{2}$$

where $\mathcal{I}(\mathbf{c})$ denotes the support of the codeword $\mathbf{c}$. Since we know from the definition of Goppa codes that,

$$\sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} = 0 \mod g(x),$$

we can compute the irreducible polynomial $g(x)$ by factoring the polynomial $c(x)$ in $\mathbb{F}_{2^m}[x]$ and choosing an irreducible factor with the weight t.

Factoring a polynomial over a finite field to irreducibles is a well-studied problem [Sho05] and can be efficiently solved by the Cantor-Zassenhaus algorithm with expected complexity $\mathcal{O}\left(l^{2+o(1)} \cdot m\right)$ or by the Berlekamp algorithm with expected complexity $\mathcal{O}\left(l^3 + l^{1+o(1)} \cdot m\right)$. For the new attack targeting the parameter sets of the Classic McEliece KEM, the polynomial degree l is bounded to a few hundred. Thus, the complexity for such polynomial factorization is low. The factorization procedure is efficiently implemented, for example, in the open-source software SageMath [The20]. We show in the next section that the task of computing $c(x)$ and factoring it to irreducibles can be performed at a negligible cost compared with the main complexity cost.

Note. To recover the polynomial $c(x)$ in Equation 2, one only needs to determine α_i where the corresponding c_i is non-zero. Thus, the weight of the codeword determines the sample complexity of recovering the irreducible polynomial $g(x)$. We therefore try to find a low-weight codeword as further explained in Section 4.3.

4 Detailed attacks

In this section, we describe a concrete instantiation of the new attack. We first propose a partial key-recovery attack to recover the irreducible polynomial $g(x)$, which is sketched in Algorithm 4. We then extend the partial key recovery to full key recovery.

Algorithm 4 The partial key-recovery side-channel attack

```
Input:  The Classic McEliece KEM parameters; the public key $\mathbf{H}''_{\mathsf{goppa}}$
Output: The partial secret key $g(x)$.
1: Find a low-weight codeword $\mathbf{c}$ with weight $w$.
2: Recover the $\alpha_i$'s for $i$ in a chosen index set $\mathcal{I}(\mathbf{c})$ with side-channel information
   of decrypting the chosen ciphertexts
3: Compute the polynomial $c(x)$ defined in Equation 2 and recover the irreducible
   polynomial $g(x)$ by factoring $c(x)$ in $\mathbb{F}_{2^m}[x]$
```

4.1 Partial key recovery

We now describe the new attack algorithm in steps. We start with producing one binary Goppa codeword with a small weight. The support of $\mathbf{c}$ is denoted by $\mathcal{I}(\mathbf{c})$. From the systematic parity check matrix $\mathbf{H}''_{\mathsf{goppa}}$, we could easily construct a generator matrix $\mathbf{G}_{\mathsf{goppa}}$ in the systematic form. Thus, the easier method to find a codeword is to randomly pick one row in the generator matrix $\mathbf{G}_{\mathsf{goppa}}$, the expected weight of the chosen codeword is $r/2 + 1$.

A more advanced approach is to use an Information Set Decoding algorithm. With Stern's algorithm as discussed in Section 2.5, we can achieve a codeword with a smaller weight.

Recovering α_i for $i \in \mathcal{I}(\mathbf{c})$ Denote $\mathbf{p_c}$ the set that $\mathbf{p_c} = \{(i, \alpha_i) : i \in \mathcal{I}(\mathbf{c})\}$. We aim to recover $\mathbf{p_c}$ with side-channel leakages. We choose an error vector $\mathbf{e}_i$ with all but the i^{th} position zero, prepare for the corresponding ciphertext, and send it to the decryption oracle. With the trained deep-learning model, we can get soft information of the corresponding α_i, i.e., a normalized vector $(l_1, l_2, \ldots, l_q) \in \mathbb{R}^q$ with $\sum l_i = 1$. A simple approach, called the naive approach, is to pick the guess with the maximized likelihood. We could design a more advanced approach, called the threshold approach, by picking a threshold τ for the obtained normalized likelihood values. If the largest likelihood value is still below the threshold τ, then we can repeat the test for $\mathbf{e}_i$ and get another normalized likelihood vector $(l'_1, l'_2, \ldots, l'_q) \in \mathbb{R}^q$. We can multiply the two vectors component-wisely and normalize the new vector. We repeat until the obtained largest likelihood value becomes larger than the threshold τ. With this method, we could recover all the α_i's for $i \in \mathcal{I}(\mathbf{c})$. The advantage of the threshold approach is that one could adaptively increase the number of decryption attempts for the obtained unreliable α_i's. Thus, if most of the positions are reliable, then the increased number of traces is limited.

Recovering the irreducible polynomial $g(x)$ As has been described in Section 3, we could compute the polynomial $c(x) \in \mathbb{F}_{2^m}[x]$ in Equation (2) if the codeword $\mathbf{c}$ and the corresponding α_i's for $i \in \mathcal{I}(\mathbf{c})$ are all determined. We recover the irreducible polynomial by factoring $c(x)$ and choosing an irreducible factor with weight t.

Last, even if we cannot uniquely determine $g(x)$, we can prepare a small list of $g(x)$'s and recover the full key for all the candidates in the list. We can find the correct one in the list since the wrong guess of $g(x)$ can be easily detected and discarded with the encryption and decryption tests.

4.2 Full key recovery

Recovering the full secret support when $n = 2^m$ After recovering the irreducible polynomial $g(x)$, if $n = 2^m$, i.e., for the parameter set kem/mceliece8192128, it is clear that the support splitting algorithm [Sen00] can be used to recover the full secret support. Since Goppa codes behave like random codes, the complexity of the support splitting algorithm is $\mathcal{O}\left(n^3\right)$, which is a small cost. We refer the interested readers to [OS09] for details.

4.2.1 Attack variant when $n < 2^m$

When $n = 2^m$, the key recovery problem is equivalent to the problem of recovering the irreducible polynomial $g(x)$ and it is beneficial to find a codeword of low weight. But this is not true in a more general case, where $n < 2^m$, since it is difficult to guess the set $\{\alpha_1, \ldots, \alpha_n\}$ and we need to design a new approach to recover the full secret support $\mathbf{p}$.

We first recover the matrix $\mathbf{P}$ that is the first $r = mt$ columns of $\mathbf{H}'_{\mathsf{goppa}}$. Assuming the partial support $(\alpha_1, \ldots, \alpha_r)$ and the irreducible polynomial $g(x)$ is known, we could compute $\mathbf{P}'$ where the element in the i^{th} row and the j^{th} column is $\alpha_j^{i-1}/g(\alpha_j) \in \mathbb{F}_{2^m}$. We then compute $\mathbf{P}$ by replacing each entry in $\mathbf{P}'$ with an m-bit column over $\mathbb{F}_2$. Then, since

$$\mathbf{P}^{-1}\mathbf{H}'_{\mathsf{goppa}} = \mathbf{H}''_{\mathsf{goppa}} = [\mathbf{I}_{mt}|\mathbf{T}],$$

one could recover $\mathbf{H}'_{\mathsf{goppa}}$ by computing $\mathbf{P}\mathbf{H}''_{\mathsf{goppa}}$.

We then build a table storing pairs $(\alpha, \mathsf{TABLE}(\alpha))$, where α runs through all the 2^m elements in $\mathbb{F}_{2^m}$. For each $\alpha \in \mathbb{F}_{2^m}$, we compute a column vector $\mathbf{K} \in \mathbb{F}_{2^m}^t$, where the i^{th} entry in $\mathbf{K}$ is $\alpha^{i-1}/g(\alpha)$. We obtain a new vector $\mathbf{K}' \in \mathbb{F}_2^{mt}$ by replacing one entry in $\mathbf{K}$ with an m-bit column over $\mathbb{F}_2$. We put $\mathsf{TABLE}(\alpha) = \mathbf{K}'$. For each column in $\mathbf{H}'_{\mathsf{goppa}}$, we can check the table and find the corresponding value of α.

Based on this new approach, we could slightly modify the attack described in Section 4.1 to make it more efficient for the parameter sets with $n < 2^m$.

Firstly, instead of minimizing the weight of the codeword, we aim to find a codeword $\mathbf{c}$ minimizing $\#\{\mathcal{I}(\mathbf{c}) \setminus \{1, \ldots, r\}\}$. An easy solution is to generate $\mathbf{G}_{\mathsf{goppa}} = [\mathbf{T}^{\mathrm{T}}|\mathbf{I}_{k\times k}]$ and to select one row in the matrix. We select the index set $\mathcal{I} = \mathcal{I}(\mathbf{c}) \cup \{1, \ldots, r\}$ and recover such $(r+1)$ α_i's with $i \in \mathcal{I}$ by observing the side-channel leakages from decrypting the chosen ciphertexts. Similar to the method described in Section 4.1, we compute $c(x)$ and

Table 2: The predicted weight of the low-weight codeword using Stern's algorithm.

	$r+1$	Estimation			≈ 2 min
		random	$\approx 2^{40}$	$\approx 2^{50}$	
kem/mceliece348864	769	385	300	282	287
kem/mceliece460896	1249	625	523	500	508
kem/mceliece6688128	1665	833	717	690	693
kem/mceliece6960119	1548	775	662	635	646
kem/mceliece8192128	1665	833	717	688	695

obtain the irreducible polynomial $g(x)$ by polynomial factorization. Last, we compute the $mt \times mt$ matrix $\mathbf{P}$, construct the secret parity-check matrix $\mathbf{H}'_{\mathsf{goppa}}$ and reconstruct the whole $\mathbf{p} = (\alpha_1, \ldots, \alpha_n)$ by checking the table $(\alpha, \mathsf{TABLE}(\alpha))$ for all $\alpha \in \mathbb{F}_{2^m}$.

Overall, the sample complexity is $(r+1)$ and the computation cost is also small. The computation of $g(x)$ by factoring $c(x)$ can be done in seconds; We build a table of size 2^m, which is small since m is only 12 or 13. The rest is dominated by the matrix multiplication of $\mathbf{PH}''_{\mathsf{goppa}}$, which is less than 2^{40} bit-operations.

4.3 Complexity analysis and verification

We conclude this section with the complexity analysis of the new algorithm and present some experimental verification. Since the profiling stage is irrelevant to the targeted public-secret keypair, we treat it as a pre-computation step.

Verifying the ISD step The weight of the found codeword $\mathbf{c}$ is a key parameter for the partial key recovery of $g(x)$ since it determines the number of α's that need to be recovered in the profiled side-channel attacking step. For the Classic McEliece KEM parameters, we compute the expected weight of the low-weight codeword with the complexity formula presented in Section 2.5 for Stern's algorithm. We show the results in the "Estimation" columns in Table 2. The column "random" shows the expected weight numbers computed by $r/2+1$, i.e., when the codeword is generated by randomly picking a row in a systematic generator matrix; the column "$\approx 2^{40}$" shows the expected weight number when the computation cost is about 2^{40} bit operations; the column "$\approx 2^{50}$" shows the case with about 2^{50} bit operations.

The complexity analysis for Stern's algorithm is well-established. One question is how to interpret the bit operation numbers as actual CPU clock cycles since modern CPUs can do more than one bit operation in a clock cycle. Since the binary Goppa codes behave like binary random linear codes, we generate parity check matrices for random linear codes with dimension same to the Classic McEliece KEM parameters and test the actual performance of Stern's algorithm. We slightly modified a C implementation [Vas05] from Vasseur that employs the AVX2 instruction set. We use eight threads in a desktop with CPU Intel(R) Core(TM) i7-10700K @3.8GHz, run the ISD algorithm for about 2 minutes, and report the obtained weights under the column "≈ 2 min" in Table 2. Thus, the numbers presented in the column "$\approx 2^{50}$" could be achieved easily on a normal desktop.

We have also performed a large instance against a random public key generated by the round-3 reference implementation of kem/mceliece8192128 submitted to NIST. Using twelve threads of the same desktop, we found a codeword of weight 690 after 103 seconds of computation, which supports the numbers claimed in Table 2. After 22858 seconds (≈ 6.3 hours), we obtained a codeword of weight 679. For a public key of kem/mceliece348864, after 47855 seconds (≈ 13 hours), we obtained a codeword of weight 273.

Verifying the factorization method Given a low-weight codeword $\mathbf{c}$, we assume that the corresponding partial secret support $\mathbf{p_c} = \{(i, \alpha_i) : i \in \mathcal{I}(\mathbf{c})\}$ has been recovered through side channels. We have verified that the procedure of computing the polynomial $c(x)$ and then factoring $c(x)$ to find $g(x)$ as the irreducible factor of weight t can be done efficiently. We implemented this procedure with the SageMath software. For instance, using a single thread in the desktop with CPU Intel(R) Core(TM) i7-10700K @3.8GHz, we performed this procedure ten times, targeting a secret Goppa code for the largest parameter set kem/mceliece8192128. We found the correct irreducible polynomial $g(x)$ of weight 128 ten times, so the empirical success probability is 100%. The average time consumed for each run is only 16.3 seconds.

The overall complexity We verified in Section 5 that when the KEM kem/mceliece348864 is implemented on our real (FPGA and ARM Cortex-M4) platforms, one decryption oracle call in the attack phase is sufficient to recover one error locator polynomial (which is equivalent to one α_i value in our setting), due to the significant leakages detected. Thus, the attack stage is quite efficient. For partial key recovery of $g(x)$, one can make the computation dominated by the initial information set decoding (ISD) step. This ISD step can be seen as a time-sample trade-off since heavier ISD computation can lead to an attack that requires fewer traces (i.e., accessing time) to the targeted device.

Thus, one can get an estimation of the sample complexity of the new partial key recovery of $g(x)$ targeting different Classic McEliece parameters, by checking the weight prediction shown in Table 2. When aiming for the full key recovery, for the KEM kem/mceliece8192128, the sample complexity is the same as that of the partial key recovery, i.e., we need 688 traces when the computation cost is bounded to 2^{50} bit operations with Stern's algorithm. For the parameter sets other than the KEM kem/mceliece8192128, we need $r+1$ traces to perform the full key recovery. The numbers $r+1$ for different parameter sets are shown in the column marked by "$r+1$" of Table 2.

5 Experimental results

In this section, we present the detailed results of our side-channel key-recovery attack against a hardware and a software implementation of Classic McEliece. To capture traces, we make use of the open-source Chipwhisperer system that consists of hardware modules and software specifically developed by NewAE for evaluating hardware security [New].

5.1 On the reference FPGA implementation

For the attack on the hardware implementation, we use the CW305 from NewAE that contains a XC7A100T2FTG256 Artix 7 FPGA. The implementation of [WSN18] is synthesized with the parameter set kem/mceliece348864 and implemented on the FPGA. To capture traces, we use the Chipwhisperer-lite (CWL) from NewAE that measures power consumption of the FPGA by the voltage drop over a shunt resistor placed inline with the supply to the FPGA. Both the FPGA and the CWL are driven by a common 5 MHz clock and the measured power is amplified by 46 dB before being digitized.

5.1.1 Leakage assessment

To evaluate whether the hardware implementation leaks sensitive side-channel information, we perform fixed-vs-random TVLA. More specifically, we want to determine if the power consumption during processing a specific $\sigma(x)$ of degree 1 differs from a random $\sigma(x)$ of degree 1. Initially, we randomly pick an element $\gamma \in \mathbb{F}_{2^m}$. Then, we construct a set of random keys $\mathcal{K}_L$ where for each $k_L \in \mathcal{K}_L$, $\mathbf{p} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ such that $\gamma = \mathbf{p}_i$ for some $i \in (1, 2, \ldots, n)$. During trace capture, we randomly decide to either capture a fixed or a random trace. For a fixed trace, we randomly select a $k_L \in \mathcal{K}_L$ and determine i such that $\gamma = \mathbf{p}_i$. In the case of a random trace, we randomly pick a $k_L \in \mathcal{K}_L$ and an integer i such

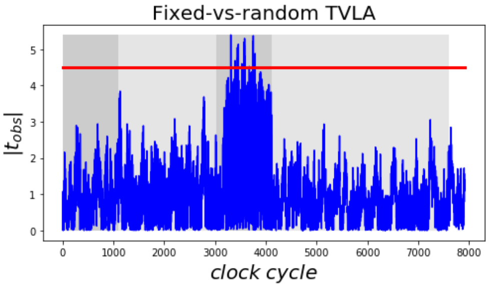

Figure 1: TVLA performed on traces of the hardware implementation where each set consists of 300 traces. The areas marked in the graph corresponds to the decryption steps; calculate $1/g^2(x)$, BM, additive FFT, and error reconstruction. The double syndrome calculation is removed from traces as this step is of non-constant time.

that $\gamma \neq \mathbf{p}_i$. In both cases, we create a plaintext $\mathbf{e}$ where the i^{th} bit $\mathbf{e}_i = 1$ and all other bits are zero. The secret key and the ciphertext $\mathbf{s}$ are then transferred to the FPGA. We then capture a trace while the FPGA runs the decryption algorithm.

Traces are then split into two sets $\mathcal{T}_0$ and $\mathcal{T}_1$, where $\mathcal{T}_0$ contains all fixed traces and $\mathcal{T}_1$ contains all random traces. Finally, we apply Welch's t-test on the two sets. We then repeat the same procedure for multiple γ's. In Fig. 1 the test statistic is shown for one of the γ's. It can be seen that, possibly exploitable, side-channel information leaks during the additive FFT step. Note that the double syndrome construction step was stripped from traces before the TVLA as the execution time of this step depends on $w_{\text{H}}(\mathbf{s})$. Thus, if this step had been kept, subsequent decryption steps would become misaligned in the captured traces, which could lead to falsely detected leakage.

5.1.2 Profile phase

In the profile phase, we only capture traces of the additive FFT step as this is the only step where we could detect leakage for our suggested attack. By focusing on the additive FFT step and using a sample rate of 1pt/clock cycle, each trace contains 1095 samples. Initially, we create a set of random keys $\mathcal{K}_p$. We capture a set of traces $\mathcal{T}_p$ with cardinality $d = \#\{\mathcal{T}_p\}$. For each $T_j \in \mathcal{T}_p$, $j = 1, 2, \ldots, d$, we pick a random $k_p \in \mathcal{K}_p$ and a random integer $i \in (1, \ldots, n)$. We capture T_j while the FPGA performs decryption of the plaintext $\mathbf{e}$ where the i^{th} bit $\mathbf{e}_i = 1$ and all other bits are zero. We label each T_j with the support element at position i of the key k_p, i.e. we label a trace with $\gamma = \alpha_i$.

We use $\mathcal{T}_p$ to train a MLP that predicts γ for each trace, i.e. the root of $\sigma(x)$. We split $\mathcal{T}_p$ into a training set $\mathcal{T}_{train}$ and validation set $\mathcal{T}_{val}$ with a ratio of 5:1. We preprocess all traces by removing the mean and scaling to unit variance before training the MLP. Parameters used for preprocessing are solely based on $\mathcal{T}_{train}$.

The architecture of our MLP is presented in Table 3. All weights in the network are initialized by sampling a uniform distribution and all biases are initially set to zero. We train the MLP by using mini-batches consisting 150 traces each. To evaluate the performance of the MLP we use the crossentropy loss and we use the Nadam optimizer with a learning rate of 10^{-3} to tune weights and biases of the network. We regularize the training by employing label smoothing with a value of 0.2. We let the training run for a maximum of 100 epochs with an early stop condition if the validation loss is not reduced within 4 epochs. For profiling we use $\#\{\mathcal{K}_p\} = 30$ and $d = 418560$.

5.1.3 Attack phase

To evaluate our approach, the trained model is used to attack a set of random keys $\mathcal{K}_a$, where $\mathcal{K}_a \cup \mathcal{K}_p = \emptyset$ and $\#\{\mathcal{K}_a\} = 30$. For each $k_a \in \mathcal{K}_a$, we collect a trace corresponding to the decryption of every possible plaintext $\mathbf{e}_i$ with only the i^{th} bit set to 1. For each trace, we also record the value of i. In total, we collect 104640 traces. Next, we employ the previously trained classifier to predict the value of each γ. Since we recorded the value of i, we know the position of the predicted γ in $\mathbf{p} = (\alpha_1, \alpha_2, \cdots, \alpha_n)$. Thus, we can predict the value of α_i. With our attack, we manage to successfully recover the complete secret support for all attacked keys. This means that the experimental prediction rate was 100%.

5.2 On an ARM Cortex-M4 implementation

For the attack on the software implementation, we use the CW308T-STM32F from NewAE which contains a STM32F415RGT6 microcontroller that is programmed with the kem/mceliece348864 implementation of [CC21]. The software implementation is built using the arm-none-eabi-gcc compiler set with optimization level -O3. We only program the microcontroller with the decryption function which is the first function called during decapsulation in [CC21]. During trace capture, the microcontroller and the CWL are driven by a common 24 MHz clock, and the measured power is amplified by 32 dB.

5.2.1 Leakage assessment

To evaluate possible leakage points for the software implementation, we use a similar approach as used for the hardware implementation. However, as the software implementation takes roughly 340 times more clock cycles to execute, the buffer of the CWL is too small to capture the complete decryption with one sample per clock cycle. We solve this by performing a piece-wise TVLA of the complete decryption. In Fig. 2 we see that the software implementation shows a large number of leakage points. Especially, we see many leakage points during the additive FFT step marked by a gray box in Fig. 2. But the TVLA also shows leakage during other parts of decryption. The detected leakage towards the end of decryption is related to the re-encryption of the recovered plaintext. The re-encryption feature is not implemented in the hardware reference design.

The software implementation makes heavy use of bit-sliced operations to speed up the decryption process where the coefficients of $\sigma(x)$ are stored in an array. Each row of the array represents a coefficient of $\sigma(x)$ and the columns represent the bits of each coefficient. During a large part of the additive FFT, operations are carried out on columns, i.e. on bits of the coefficients rather than the whole coefficients. During our attack, we make sure that $\sigma(x)$ has a single root and thus the individual bits of the root should affect the power consumption at different times during the FFT. To investigate this, we captured 34880 traces while the additive FFT was fed by a random degree 1 polynomial. For each bit i of the 12 bits representing elements of $\mathbb{F}_{2^m}$ we assigned traces to one out of

Table 3: The architecture of the MLP we use to construct a classifier to attack the hardware reference implementation of Classic McEliece.

Layer type	(input, output) shape	Activation	# Parameters
Dense	(1095, 1095)	ReLu	1200120
Batch Normalization	(1095, 1095)		4380
Dense	(1095, 2048)	ReLu	2244608
Batch Normalization	(2048, 2048)		8192
Dense	(2048, 4096)	SoftMax	8540160

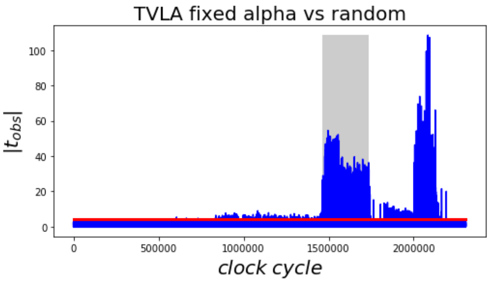

Figure 2: TVLA of the software implementation. The gray box corresponds to the additive FFT evaluation of $\sigma(x)$.

two sets depending on the value of bit i. Fig. 3 shows the difference between the means for the two sets for each bit position i. The left part of Fig. 3 shows the difference for the complete additive FFT where the black dashed lines show where different steps of the FFT correspond to Lines 2-5 in Listing 2. In the right part of Fig. 3, the second step of the FFT corresponds to the loops at Lines 3-11 in Listing 3. In this part, all the degree 1 polynomials from the radix conversion step are evaluated. As seen the power consumption at different times depends on bits of the $\sigma(x)$ that we fed to the FFT. This can be explained by the assignment at Line 8 in Listing 3 where the variable `vv` gets assigned the value 0 or 1 depending on a single bit of the input to the function.

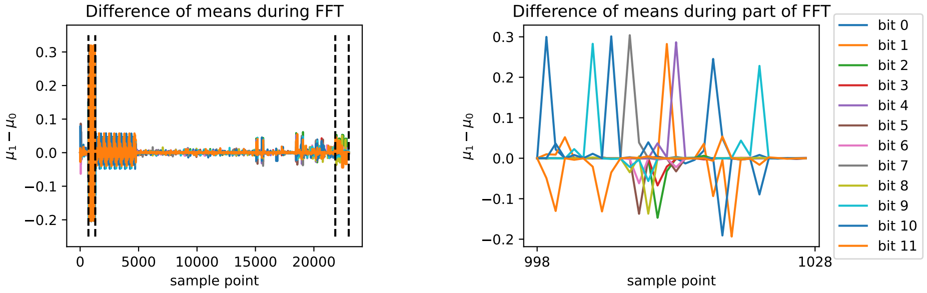

Figure 3: Difference in power consumtion of the Cortex M4 during the additive FFT while diffrent bits of the coefficient of the degree 1 $\sigma(x)$ are set to 0 or 1. The left figure shows the complete additive FFT while the right figure shows a small part of the second step in the additive FFT.

```
1   void fft_p64_v4096_u32(uint32_t values [][GFBITS], uint32_t poly [][GFBITS
      ]){
2     radix_conversions_u32(poly);
3     broadcast_64x_32(values,poly);
4     butterflies_6_u32(values);
5     add_x64_u32(values);
6   }
```

Listing 2: The additive FFT function in [CC21]

```
1 static void broadcast_64x_32(uint32_t out_32[][ GFBITS ], uint32_t in_32[][
      GFBITS ]){
2   ...
3   for (j = 0; j < 64; j++) {
4     pos = reversal[j]>>5;
5     idx = reversal[j]&0x1f;
6     for (i = 0; i < GFBITS; i++)
7     {
```

Table 4: The architecture of the MLP we use to construct a classifier to attack the software implementation.

Layer type	(input, output) shape	Activation	# Parameters
Dense	(22988, 22988)	ReLu	528471132
Batch Normalization	(22988, 22988)		91952
Dense	(4096, 4096)	ReLu	94162944
Batch Normalization	(22988, 22988)		16384
Dense	(4096, 4096)	SoftMax	16781312

```
        uint32_t vv = (in_32[pos][i] >> idx )&1;
        vv = -vv;
        out_32[j*2][i] = vv;
        out_32[j*2+1][i] = vv;
          ...
}
```

Listing 3: Part of the second step of the additive function in [CC21]

5.2.2 Profile phase

In the profile phase we only focus on the additive FFT step of the algorithm. As the duration of the additive FFT part is too long to be sampled with the CWL at every clock cycle there are two options. We could capture the FFT step in intervals with one sample per clock cycle, or we could capture the complete FFT step with a lower sample rate. As the time and memory requirements of the subsequent neural network training would be higher for the first capture procedure, we decide to test if the later option works. Therefore, we reduce the sample rate to 1pt/12$^{\text{th}}$ microcontroller clock cycle. This results in traces consisting of 22988 sample points. Apart from the number of sample points, we use the same procedure to capture traces as for the hardware implementation. We train the MLP given in table 4 with the same initialization, optimizer and regularization as for the profiling on the hardware implementation. We let the training run for a maximum of 100 epochs with an early stop condition if the validation loss is not improved within 3 epochs. For profiling of the software classifier we use $\#\{\mathcal{K}_p\} = 10$ and $d = 34880$.

5.2.3 Attack phase

We evaluate our classifier by attacking a set of random keys $\mathcal{K}_a$, where $\mathcal{K}_a \cup \mathcal{K}_p = \emptyset$ and $\#\{\mathcal{K}_a\} = 90$. We use the same procedure to capture traces as for the hardware implementation, i.e. for each $k_a \in \mathcal{K}_a$ we capture 3488 traces which are then fed to the classifier. Thus, in total we collect 313920 traces. With our attack against the software implementation, we manage to successfully recover the complete secret support for all the attacked keys. Again, the experimental prediction rate was 100%.

5.3 Alternative approach

Since our neural network classifiers performed very well, we also tested if some simple statistical method could be used instead of the neural network. Here we describe the attack on the software implementation of Classic McEliece but we also performed the same type of attack on the hardware implementation.

5.3.1 Profile phase

We capture a set of traces $\mathcal{T}_p$ corresponding to Line 3 in Listing 3, i.e. the evaluation of the degree 1 polynomials outputted by the radix conversion function. For each $T_j \in \mathcal{T}_p$, $j = 1, 2, \ldots, d$, we pick a key k_p from a set of random keys $\mathcal{K}_p$ and a random integer $i \in (1, \ldots, n)$. We capture T_j while the microcontroller performs decryption of the plaintext $\mathbf{e}$ where the i^{th} bit $\mathbf{e}_i = 1$ and all other bits are zero. We label each T_j with the known support element at position i of the key k_p, i.e. we label a trace with the field element α_i. Next, we split the set $\mathcal{T}_p$ into sets $\mathcal{A}_k$, $k = 0, 1, \ldots, 4095$, based on the label of the traces. For each $\mathcal{A}_k$, we calculate the mean $\mu_{\mathcal{A}_k}$ of the traces in the set. During the profile phase, we use $\#\{\mathcal{T}_p\} = 313920$ and $\#\{\mathcal{K}_p\} = 90$.

5.3.2 Attack phase

First we load a random key $k_a \in \mathcal{K}_a$ where $\mathcal{K}_a \cup \mathcal{K}_p = \emptyset$. Next, we collect traces T_i, $i = 1, 2, \ldots, 3488$, corresponding to the decryption of every possible plaintext $\mathbf{e}_i$ with only the i^{th} bit set to 1. For each trace, we record the value of i and for each key k_a we collect 3488 traces. To predict the secret support we compare each trace T_i with all the $\mu_{\mathcal{A}_k}$ from the profile phase and assign

$$\alpha_i = \arg\min_k(|T_i - \mu_{\mathcal{A}_k}|). \tag{3}$$

Since we recorded the value i this gives us a prediction of the secret support element at position i, i.e. α_i. We evaluate our distinguisher on 10 keys where we manage to fully recover the secret support of each key. This means that the experimental success rate was 100%.

We performed the same type of attack on the hardware implementation but in this case the success rate was 0.05 % which is only slightly better than the random guess success rate of 0.024 %.

6 Discussions and concluding remarks

In this paper, we have presented the first key-recovery side-channel attack on Classic McEliece, a KEM finalist in the NIST Post-quantum Cryptography Standardization Project. We have identified a general vulnerability in the decryption algorithm design that the additive FFT procedure for evaluating a polynomial at many points is deterministic for a fixed input error locator polynomial. We then designed special ciphertexts generated by error vectors where the Hamming weight is 1 and captured traces of the decryption of such ciphertexts. Since the error weight is only 1, there are only $q = 2^m$ possible error locator polynomials. We then used machine-learning algorithms to recover the secret error locator polynomial, which reveals one entry in the secret support $\mathbf{p} = (\alpha_1, \ldots, \alpha_n)$. We have also designed new algorithms to recover the partial key $g(x)$ and the full secret key, based on the side-channel leakages.

We implemented and measured the new attack in real FPGA and ARM Cortex-M4 platforms. We have a perfect recovery for recovering the secret α_i with one trace, i.e., the empirical success probability is 100% with the naive approach. Note that in a noisier platform, we can use the threshold approach to adaptively send more decryption traces for the unreliable α_i's.

The sample complexity of the partial key recovery attack depends on the computation limit of the first ISD step in the proposed attack. We could achieve 273 traces for kem/mceliece348864 and 679 traces for kem/mceliece8192128. For the case $n = q$, i.e. kem/mceliece8192128, a partial key recovery is equivalent to a full key recovery with the help of the support splitting algorithm.

The other four parameter sets with $n \neq q$ provide more security against this side-channel attack. We designed a variant of the full key recovery attack with sample complexity of $r+1$ traces, which is 769 for the KEM kem/mceliece348864. A future work is to study better algorithms to efficiently recover the full secret support from the recovered $g(x)$ when $n \neq q$.

On the one hand, the Classic McEliece KEM still requires more traces compared with the recent results in attacking lattice-based primitives. For instance, in [NDGJ21], the secret key of a masked Saber version could be recovered with around 20 traces. On the other hand, this work shows that protections like masking are necessary, though they may hurt the performance of the scheme. Thus, research on efficiently protected implementations of Classic McEliece should be prioritized.

Our key-recovery attack requires slightly more traces than the message-recovery EM attack reported at Asiacrypt 2020 [LNPS20]. But, we highlight that key-recovery attacks are much stronger than message-recovery attacks. Also, we could greatly reduce the sample complexity in a theoretical manner. One approach is to send a ciphertext obtained from a plaintext of higher weight (of e.g. weight 2). For $\mathbf{e}$ with $w_{\mathrm{H}}(\mathbf{e}) = 2$, there are $q(q-1)/2$ possible error locator polynomials to profile. The number is about 2^{23} for the kem/mceliece-348864 KEM and about 2^{25} for the other parameter sets. Then, assuming that one error locator polynomial can be recovered by one decryption oracle call, each time we can determine two α_i's. The overall sample complexity can be reduced by a factor of 2. This improved attack has a high implementation complexity, but is still doable; we leave it for future work. Last, an interesting future direction is to improve the current attack without a significant increase in the profiling complexity.

The presented results are limited to the traditional setting in power analysis. Extensions to EM analysis seem plausible, since the accuracy in the measurements is very high, but not investigated in this work. Devices facing threats from power analysis would have to implement countermeasures. A low-order masked hardware implementation may be quite safe, but still it can severely hurt the performance. We have not found such a protected implementation. Low-order masked software implementations may on the other hand be attacked similar to attacks on lattice-based and code-based KEMs. An adversary using machine learning can perform higher-order attacks since the input traces contain information related to all the shares [BDK+20, NDGJ21, BDH+21, BGR+21, ABH+22].

Acknowledgements

We would like to thank Prof. Kenny Paterson for his shepherding care and the reviewers for their valuable feedback. The work presented in this paper has been partly funded by the Swedish Research Council (Grants No. 2019-04166 and No. 2021-04602), by the Swedish Civil Contingencies Agency (Grants No. 2020-11632), by the Swedish Foundation for Strategic Research (Grant No. RIT17-0005), and by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[AAB+20] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and Jurjen Bos. HQC. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[ABB+20] 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, and Santosh Ghosh. BIKE. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[ABC+20] 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, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[ABH+22] Melissa Azouaoui, Olivier Bronchain, Clément Hoffmann, Yulia Kuzovkova, Tobias Schneider, and François-Xavier Standaert. Systematic study of decryption and re-encryption leakage: the case of kyber. Cryptology ePrint Archive, Report 2022/036, 2022. https://eprint.iacr.org/2022/036.

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

[AR21] Amund Askeland and Sondre Rønjom. A side-channel assisted attack on NTRU. Cryptology ePrint Archive, Report 2021/790, 2021. https://eprint.iacr.org/2021/790.

[BCS13] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 250–272, Santa Barbara, CA, USA, August 20–23, 2013. Springer, Heidelberg, Germany.

[BDH+21] Shivam Bhasin, Jan-Pieter D'Anvers, Daniel Heinz, Thomas Pöppelmann, and Michiel Van Beirendonck. Attacking and defending masked polynomial comparison. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):334–359, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8977.

[BDK+20] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel resistant implementation of SABER. Cryptology ePrint Archive, Report 2020/733, 2020. https://eprint.iacr.org/2020/733.

[BGG+15] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications*, pages 64–81, Cham, 2015. Springer International Publishing.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8874, pages 326–343. Springer, Berlin, Heidelberg, 2014.

[BGR$^+$21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking kyber: First- and higher-order implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):173–214, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9064.

[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*, volume 7237 of *Lecture Notes in Computer Science*, pages 520–536, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[CC21] Ming-Shing Chen and Tung Chou. Classic mceliece on the ARM cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):125–148, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8970.

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

[CDCG22] Brice Colombier, Vlad-Florin Dragoi, Pierre-Louis Cayrel, and Vincent Grosso. Message-recovery profiled side-channel attack on the classic mceliece cryptosystem. Cryptology ePrint Archive, Report 2022/125, 2022. https://ia.cr/2022/125.

[CDH$^+$20] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, Tsunekazu Saito, Takashi Yamakawa, and Keita Xagawa. NTRU. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68, Taipei, Taiwan, September 25–28, 2017. Springer, Heidelberg, Germany.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[DKR$^+$20] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[FHS+17] Tomás Fabsic, Viliam Hromada, Paul Stankovski, Pavol Zajac, Qian Guo, and Thomas Johansson. A reaction attack on the QC-LDPC McEliece cryptosystem. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 51–68, Utrecht, The Netherlands, June 26–28 2017. Springer, Heidelberg, Germany.

[Ger] German Federal Office for Information Security (BSI). BSI TR-02102-1: Cryptographic Mechanisms: Recommendations and Key Lengths. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf. Accessed: 2022-02-15.

[GHJ+21] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in hqc and bike. Cryptology ePrint Archive, Report 2021/1485, 2021. https://ia.cr/2021/1485.

[GJ20] Qian Guo and Thomas Johansson. A new decryption failure attack against HQC. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 353–382, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for sidechannel resistance validation. In *NIST non-invasive attack testing workshop*, 2011.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 359–386, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.

[GJS16] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 789–815, Hanoi, Vietnam, December 4–8, 2016. Springer, Heidelberg, Germany.

[GJW19] Qian Guo, Thomas Johansson, and Paul Stankovski Wagner. A key recovery reaction attack on QC-MDPC. *IEEE Trans. Information Theory*, 65(3):1845–1861, 2019.

[HG13] Stefan Heyse and Tim Güneysu. Code-based cryptography on reconfigurable hardware: tweaking niederreiter encryption for performance. *Journal of Cryptographic Engineering*, 3(1):29–43, April 2013.

[HMP10] Stefan Heyse, Amir Moradi, and Christof Paar. Practical power analysis attacks on software implementations of McEliece. In Nicolas Sendrier, editor, *The Third International Workshop on Post-Quantum Cryptography, PQCRYPTO 2010*, pages 108–125, Darmstadt, Germany, May 25–28 2010. Springer, Heidelberg, Germany.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology*

– *CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.

[KPH$^+$19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise: Unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):148–179, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8292.

[LB88] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *Advances in Cryptology – EUROCRYPT'88*, volume 330 of *Lecture Notes in Computer Science*, pages 275–280, Davos, Switzerland, May 25–27, 1988. Springer, Heidelberg, Germany.

[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.

[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, Part I*, volume 12491 of *Lecture Notes in Computer Science*, pages 881–910, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

[LPB$^+$15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 20–33, Cham, 2015. Springer International Publishing.

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

[Mag19] Houssem Maghrebi. Deep learning based side channel attacks in practice. *IACR Cryptol. ePrint Arch.*, 2019:578, 2019.

[Mas69] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theory*, 15(1):122–127, 1969.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic coding theory. *JPL DSN Progress Reports*, 44:114–116, 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, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 107–124, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[MO15] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 203–228, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[MS78] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes.* North-holland Publishing Company, 2nd edition, 1978.

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

[NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A side-channel attack on a masked IND-CCA secure saber KEM implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(4):676–707, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9079.

[New] NewAE Technology Inc. Chipwhisperer. https://newae.com/chipwhisperer.

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

[NIS] NIST Post-Quantum Cryptography Standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization. Accessed: 2018-09-24.

[OS09] Raphael Overbeck and Nicolas Sendrier. *Code-based cryptography*, pages 95–145. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[PCBP21] Guilherme Perin, Łukasz Chmielewski, Lejla Batina, and Stjepan Picek. Keep it unsupervised: Horizontal attacks meet deep learning. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):343–372, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8737.

[PPM$^+$21] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. SoK: Deep learning-based physical side-channel analysis. Cryptology ePrint Archive, Report 2021/1092, 2021. https://eprint.iacr.org/2021/1092.

[Pra62] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Trans. Information Theory*, 8(5):5–9, 1962.

[REB$^+$22] Prasanna Ravi, Martianus Frederic Ezerman, Shivam Bhasin, Anupam Chattopadhyay, and Sujoy Sinha Roy. Will you cross the threshold for me? generic side-channel assisted chosen-ciphertext attacks on ntru-based kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):722–761, 2022.

[RJJ$^+$19] Prasanna Ravi, Bernhard Jungk, Dirmanto Jap, Zakaria Najm, and Shivam Bhasin. Feature Selection Methods for Non-Profiled Side-Channel Attacks on ECC. In *International Conference on Digital Signal Processing, DSP*, volume 2018-Novem. Institute of Electrical and Electronics Engineers Inc., jan 2019.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):307–335, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8592.

[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/round-3-submissions`.

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

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

[Sho05] Victor Shoup. *A Computational Introduction to Number Theory and Algebra.* Cambridge University Press, USA, 2005.

[SKL+20] Bo-Yeon Sim, Jihoon Kwon, Joohee Lee, Il-Ju Kim, Tae-Ho Lee, Jaeseung Han, Hyojin Yoon, Jihoon Cho, and Dong-Guk Han. Single-trace attacks on message encoding in lattice-based kems. *IEEE Access*, 8:183175–183191, 2020.

[SS92] V. M. Sidelnikov and S. O. Shestakov. On insecurity of cryptosystems based on generalized reed-solomon codes. *Discrete Mathematics and Applications*, 2(4):439–444, 1992.

[SSMS10] Abdulhadi Shoufan, Falko Strenzke, H. Gregor Molter, and Marc Stöttinger. A timing attack against Patterson algorithm in the McEliece PKC. In Donghoon Lee and Seokhie Hong, editors, *ICISC 09: 12th International Conference on Information Security and Cryptology*, volume 5984 of *Lecture Notes in Computer Science*, pages 161–175, Seoul, Korea, December 2–4, 2010. Springer, Heidelberg, Germany.

[Ste88] Jacques Stern. A method for finding codewords of small weight. In Gérard D. Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications, 3rd International Colloquium, Toulon, France, November 2-4, 1988, Proceedings*, volume 388 of *Lecture Notes in Computer Science*, pages 106–113. Springer, 1988.

[STM+08] Falko Strenzke, Erik Tews, H. Gregor Molter, Raphael Overbeck, and Abdulhadi Shoufan. Side channels in the McEliece PKC. In Johannes Buchmann and Jintai Ding, editors, *Post-quantum cryptography, second international workshop, PQCRYPTO 2008*, pages 216–229, Cincinnati, Ohio, United States, October 17–19 2008. Springer, Heidelberg, Germany.

[The20] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.0)*, 2020. `https://www.sagemath.org`.

[Tim18] Benjamin Timon. Non-profiled deep learning-based side-channel attacks. Cryptology ePrint Archive, Report 2018/196, 2018. `https://eprint.iacr.org/2018/196`.

[UXT+22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/em analysis on post-quantum kems. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):296–322, 2022.

[Vas05] Valentin Vasseur. An implementation of dumer's algorithm for information set decoding to solve some challenges on the code-based challenges webpage, Accessed:2022-02-05. https://github.com/vvasseur/isd.

[WSN17] Wen Wang, Jakub Szefer, and Ruben Niederhagen. FPGA-based key generator for the niederreiter cryptosystem using binary goppa codes. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 253–274, Taipei, Taiwan, September 25–28, 2017. Springer, Heidelberg, Germany.

[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, Fort Lauderdale, Florida, United States, April 9–11 2018. Springer, Heidelberg, Germany.

[XPRO20] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, and David Oswald. Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. Cryptology ePrint Archive, Report 2020/912, 2020. https://eprint.iacr.org/2020/912.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8391.