Message-recovery Horizontal Correlation Attack on *Classic McEliece*

Brice Colombier[1][0000−0002−6028−3028], Vincent Grosso[1][0000−0002−3874−7527], Pierre-Louis Cayrel[1][0000−0002−6708−868X], and Vlad-Florin Drăgoi[2,3][0000−0002−8673−9097]

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

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

Abstract. As the technical feasibility of a quantum computer becomes more and more likely, post-quantum cryptography algorithms are receiving particular attention in recent years. Among them, code-based cryptosystems were first considered unsuited for hardware and embedded software implementations because of their very large key sizes. However, recent work has shown that such implementations are practical, which also makes them susceptible to physical attacks. In this article, we propose a horizontal correlation attack on the *Classic McEliece* cryptosystem, more precisely on the matrix-vector multiplication over $\mathbb{F}_2$ that computes the shared key in the encapsulation process. The attack is applicable in the broader context of Niederreiter-like code-based cryptosystems and is independent of the code structure, *i.e.* it does not need to exploit any particular structure in the parity check matrix. Instead, we take advantage of the constant time property of the matrix-vector multiplication over $\mathbb{F}_2$. We extend the feasibility of the basic attack by leveraging information-set decoding methods and carry it out successfully on the reference embedded software implementation. Interestingly, we highlight that implementation choices, like the word size or the compilation options, play a crucial role in the attack success, and even contradict the theoretical analysis.

1 Introduction

There is almost no day without a *Quantum Computing* breaking news. During the last five years the evolution of quantum technologies went from creating and improving qubits, investigating both quantitative as well as qualitative aspects [36,44,26], to practical simulated models of connecting qubits into quantum networks [43,25], all in a common goal of achieving a scalable quantum computer.

From private companies to public organizations/governments announcing new quantum computing and security acts, all converge to this major technology challenge.The US Public Law No. 117-260 from 2022, also known as the *Quantum Computing Cybersecurity Preparedness Act* encourages the migration of information technology systems to quantum-resistant cryptography. Also, the European Union Agency for Cybersecurity urges all private data sensitive technologies to adopt quantum resistant cryptographic solutions [7].

The post-quantum cryptography standardization process initiated by NIST in 2016 aims at standardizing cryptography algorithms whose security is not threatened by the existence of a quantum computer of sufficient capacity. Two categories of algorithms are considered, namely digital signatures and key encapsulation mechanisms (KEMs). One KEM proposal under consideration in the fourth round of the standardization process is the *Classic McEliece* cryptosystem which is based on error-correcting codes [1] and its security relies on the hardness of the binary syndrome decoding problem.

While the first rounds of the standardization process focused on performing cryptanalysis of the algorithms, NIST explicitly stated that time has now come for more hardware implementations, and in particular side-channel resistant implementations. In this regard, it is of prime importance to evaluate the susceptibility of the implementations of these algorithms to physical attacks in general and to side-channel attacks in particular. It is also one of the main tracks followed by the NIST PQC seminar,[4] where the side-channel analysis topic is recurrent [39,37].

Even before being considered a strong candidate in the post-quantum cryptography race, code-based cryptosystems were subject to side-channel attacks [12,22,31,2,13]. The first round of the NIST post-quantum standardization process ended with the last documentation updates on June 12th, 2019. It was that moment that triggered even more specific side-channel attacks, *i.e.* oriented towards the *Classic McEliece* KEM. In a series of articles, either the security of session key or of the private key was successfully broken [24,11,20,19].

In this work, we focus on the core operation of the encapsulation process in the *Classic McEliece* cryptosystem, namely the matrix-vector multiplication over $\mathbb{F}_2$ between the public-key, *i.e.* the parity-check matrix of the error-correcting code, and a random vector of fixed, low Hamming weight. This operation has been targeted before by physical attacks, for instance by laser fault injection [11] or a profiled side-channel attack [17]. In both of these works, the value of the integer syndrome is exploited to recover the secret error vector. This breaks the security of the KEM and allows to recover the shared key, since the secret error vector is used as seed to derive the shared secret key.

Conversely here, we recover the secret error vector by performing an *unprofiled* horizontal correlation attack on the matrix-vector multiplication over $\mathbb{F}_2$. This attack directly relies on the constant-time property of the implementation of the matrix-vector multiplication. We show that a correlation exists between

[4] `https://csrc.nist.gov/Projects/post-quantum-cryptography/workshops-and-timeline/pqc-seminars`

the columns of the parity-check matrix and the Hamming distance leakage during the execution. This allows to identify which columns of the parity-check matrix are involved in the syndrome computation and reveals the positions of the errors in the secret error vector.

Contributions

This article makes the following contributions:

- We propose the first unprofiled attack on the *Classic McEliece* cryptosystem. This attack uses a single side-channel trace and allows to recover the shared secret key of the KEM. Since it targets the matrix-vector multiplication step, it is applicable to any Niederreiter-like cryptosystem,
- We evaluate the attack success with respect to important implementation choices, namely the size of the words used for the implementation of the matrix-vector multiplication over $\mathbb{F}_2$ and the optimization options passed to the compiler. In particular, we highlight a discrepancy between the success rate of the attack in a simulated setting and the observations made on real side-channel traces.

Organization

This article is organized as follows. Section 2 presents the *Classic McEliece* cryptosystem before reviewing existing side-channel attacks that apply against it. Section 3 describes the proposed horizontal correlation side-channel attack. Section 4 provides experimental results, both with simulated and real side-channel traces, for various parameters. In Section 5, we discuss the influence of the implementation choices on the attack success rate, before concluding in Section 6.

2 Related work

2.1 Notations

The following notations are used in this article. Sets are written as calligraphic uppercase letters, for example $\mathcal{S}$. Matrices are written in uppercase bold letters, for example $\mathbf{M}$. Vectors are written in lowercase bold letters, for example $\mathbf{v}$. The identity matrix of size n is written as $\mathbf{I}_n$. The j^{th} column of a matrix $\mathbf{M}$ is written as $\mathbf{M}_{[:,j]}$. The entry on the j^{th} column of the i^{th} row of a matrix $\mathbf{M}$ is written as $\mathbf{M}_{[i,j]}$. The i^{th} entry of a vector $\mathbf{v}$ is written as $\mathbf{v}_i$. The Hamming weight of a binary vector $\mathbf{v}$, *i.e.* the number of non-zero coordinates, is written as $\mathrm{HW}(\mathbf{v})$. The Hamming distance between two vectors $\mathbf{u}$ and $\mathbf{v}$ is written as $\mathrm{HD}(\mathbf{u}, \mathbf{v})$. The bitwise logical AND between two vectors $\mathbf{u}$ and $\mathbf{v}$ is written as $\mathbf{u} \wedge \mathbf{v}$ and the logical exclusive-OR is written as $\mathbf{u} \oplus \mathbf{v}$.

2.2 The *Classic McEliece* cryptosystem

The *Classic McEliece* cryptosystem is a candidate to the NIST post-quantum cryptography standardization process [1]. After going through the first three rounds, it has been selected as a fourth round candidate on July 5th, 2022. As a KEM, its role is to allow for secure transfer of a shared secret key. It is split into three functions: key generation, encapsulation and decapsulation. During key generation, a public key $\mathbf{H}_{\text{pub}} = (\mathbf{I}_{n-k}|\mathbf{T})$ and its associate private key $\mathbf{G}_{\text{priv}}$ are generated. The encapsulation mechanism take as input the public key and outputs the shared secret key $\mathbf{k}$ that will be used in the next exchange and an encapsulated value $\mathbf{s}$ sent to the second party. The second party runs the decapsulation using its private key $\mathbf{G}_{\text{priv}}$ and the encapsulated message $\mathbf{s}$ to derive the shared secret key $\mathbf{k}$. We focus on the encapsulation step here.

***Classic McEliece* encapsulation** We target the encapsulation function of the *Classic McEliece* cryptosystem, described in Algorithm 1 and more precisely the ENCODE subroutine, shown on line 3 in Algorithm 1 and detailed in Algorithm 2. The specific operation we target with the proposed attack is the matrix-vector multiplication over $\mathbb{F}_2$ between the parity-check matrix and the random error vector. This operation is essentially the encryption step of the Niederreiter cryptosystem [32]. In this setting, a message is first encoded into a constant-weight vector before being multiplied by the parity-check matrix of a binary Goppa code. The security of this construct relies on the $\mathcal{NP}$-hardness of the binary syndrome decoding problem [6]: knowing $\mathbf{H}_{\text{pub}}$ and $\mathbf{s}$, recovering $\mathbf{e}$ is hard.

Algorithm 1 *Classic McEliece* encapsulation [1]

1: **function** ENCAPS($\mathbf{T}$)
2: Draw a random vector $\mathbf{e} \in \mathbb{F}_2^n$ with $\text{HW}(\mathbf{e}) = t$.
3: Compute $\mathbf{c} \leftarrow$ENCODE($\mathbf{e}, \mathbf{T}$)
4: Compute $\mathbf{k} \leftarrow \mathsf{H}(1 \parallel \mathbf{e} \parallel \mathbf{c})$ ▷ session key (shared secret)
5: **return** $(\mathbf{c}, \mathbf{k})$

Algorithm 2 *Classic McEliece* encoding subroutine [1]

1: **function** ENCODE($\mathbf{e}, \mathbf{T}$)
2: Define $\mathbf{H}_{\text{pub}} \leftarrow (\mathbf{I}_{n-k}|\mathbf{T})$
3: Compute $\mathbf{s} \leftarrow \mathbf{H}_{\text{pub}}.\mathbf{e}$
4: **return** $\mathbf{s}$

Embedded software implementations For years, embedded software implementations of the Niederreiter cryptosystem were deemed impractical because

of the high memory requirements for the public key storage. The first implementation on an 8-bit microcontroller was done by Heyse in 2010 [21], but with relatively low-security parameters: $n = 2048$, $k = 1751$ and $t = 27$. These values must be compared with the ones of the *Classic McEliece* submission provided in Table 1, that lead to even larger keys. Implementation challenges of code-based cryptosystems on embedded platforms are discussed in [9].

The selection of *Classic McEliece* as a candidate for the last rounds of the NIST post-quantum cryptography standardization process sparked a renewed interest for embedded software implementations. In [38], the public key is not stored but retrieved from the private key, following a streaming approach which had been described before in [42]. However, with the memory capacity of embedded devices increasing continuously, it was only a matter of time before the public key could be fully stored in the Flash memory. This happened in 2021, when Chen and Chou proposed an implementation of the *Classic McEliece* cryptosystem on an ARM Cortex-M4 target, along with many other optimizations [14].

2.3 Side-channel attacks

Side-channel attacks exploit the information leakage that occurs when an algorithm is being executed on a physical device. Profiled attacks require access to an open device for a preliminary training step, during which the attacker can freely set the secret values and perform side-channel measurements. A classifier is trained with this data and used to attack a closed device and recover the secret values. Conversely, an *unprofiled* attack does not require a training step, and is therefore more readily applicable. Well-known examples of unprofiled attacks are the differential power analysis (DPA) [23] and the correlation power analysis (CPA) [10].

Horizontal attacks A horizontal attack, as published by Walter [45] originally, consists in observing several intermediate values inside a single side-channel trace to extract secret information from it. This is in contrast with vertical attacks, of which the DPA and the CPA are examples, where a single intermediate value is repeatedly observed over several side-channel traces.

In [45], a DPA-style attack on the modular exponentiation algorithm used in the RSA cryptosystem is carried out, allowing the attacker to recover the secret exponent. This attack was later improved by Clavier *et al.* [16]. By using correlation, authors were able to distinguish a squaring from a multiplication by identifying if intermediate values were involved in a computation or not. We follow a similar approach in this article, essentially identifying which columns of the parity-check matrix are involved in the computation.

Side-channel attacks on the Niederreiter / *Classic McEliece* cryptosystems We focus here on side-channel attacks that aim at recovering the secret message in the Niederreiter cryptosystem or equivalently the secret random vector in the *Classic McEliece* cryptosystem. It is straightforward to see

Table 1: *Classic McEliece* parameters

Parameters set	`348864`	`460896`	`6688128`	`8192128`
n	3488	4608	6688	8192
m	12	13	13	13
t	64	96	128	128
$k = n - mt$	2720	3360	5024	6528
$n - k = mt$	768	1248	1664	1664

that if one is capable of recovering the secret random vector $\mathbf{e}$, then the encapsulation step is effectively deterministic and broken. As a consequence, the attacker has access to the shared secret key, since all other operations are deterministic and the $\mathbf{H}_{\text{pub}}$ matrix is public.

In [24], authors attack the constant-time Berlekamp-Massey decoding algorithm used in the decapsulation step. This is done on the reference hardware implementation of *Classic McEliece*. They adapt the attack from [40] and identify error positions by repeatedly adding columns of the parity-check matrix to the syndrome, determining which ones correspond to a decoding failure. While the iterative chunking strategy they propose reduces the number of queries, the number of side-channel traces needed ranges from 334 to 654 for the *Classic McEliece* parameters. In contrast, the attack we propose succeeds with a single side-channel trace.

The approach followed in [17] is entirely different and targets the encapsulation step instead, in particular the matrix-vector multiplication over $\mathbb{F}_2$ used for the syndrome computation. It consists in deriving an integer syndrome from side-channel measurements, instead of the binary syndrome. Although this integer syndrome might be slightly incorrect in some positions, because of noise in the side-channel measurements and errors in the integer syndrome derivation, it is usually enough to recover the error positions using a distinguisher based on the dot product. This attack has the advantage of requiring a single side-channel measurement in the attack phase. However, being in the profiled attack setting, an open device is still required for the profiling phase. An improvement of the attack resistance to noise in the profiled setting was recently published by Grosso *et al.* [19] thanks to a t-test based attack method.

In [13], a horizontal attack is mounted not on the Niederreiter cryptosystem but on the McEliece cryptosystem, specifically on its variant that uses quasi-cyclic moderate-density parity-check (QC-MDPC) codes. This horizontal attack does not attack the syndrome computation but the key rotation, which is inherent to the quasi-cyclic property. Since the attack targets a hardware implementation, the registers which are overwritten during the key rotation step have a very distinctive leakage, which can be exploited to recover the key. Conversely, the attack we propose is more general since it does not exploit the structure of the matrix.

Guo *et al.* proposed a profiled key-recovery attack on the *Classic McEliece* cryptosystem [20]. Targetting the decapsulation step, it recovers the private key using between 300 and 800 side-channel traces. Since it targets the private key, *i.e.* the long-term secret, it is definitely a more serious threat than message-recovery attacks, which recover the shared secret key, *i.e.* the short-term secret. However, their attack requires specifically crafted ciphertexts, which correspond to single-bit error vectors. Arguably, these very specific error patterns could be easily detected during the decapsulation step and a countermeasure based on an early abort could prevent the attack.

It is worth noting that other post-quantum cryptosystems, based on lattices for instance, have been the target of horizontal side-channel attacks as well, as already highlighted in previous works [4,3].

In this work, we target the syndrome computation in the encapsulation step, but propose an *unprofiled* attack instead. Therefore, before outlining the attack procedure, we describe how the syndrome computation is usually implemented.

2.4 Software implementations of the syndrome computation

The target operation of the proposed attack is the syndrome computation, which is a matrix-vector multiplication over $\mathbb{F}_2$, as given in Equation (1). This operation is the second step of the encoding routine in the encapsulation process (see line 3 in Algorithm 2), which is called after a uniform random constant-weight vector $\mathbf{e}$ has been drawn.

$$\mathbf{s} = \mathbf{H}_{\text{pub}}.\mathbf{e} \tag{1}$$

Schoolbook implementation The matrix-vector multiplication over $\mathbb{F}_2$ is implemented in software by iterating over the matrix rows and columns and performing a logical AND operation between the matrix and the vector entries, while accumulating the result on the syndrome entry by a logical XOR operation. We refer to this matrix-vector multiplication method as the *schoolbook* method. This is described in Algorithm 3, where an $(n-k) \times n$ matrix $\mathbf{M}$ is multiplied by an n-bit vector $\mathbf{v}$ to obtain an $(n-k)$-bit syndrome $\mathbf{s}$.

Algorithm 3 Schoolbook matrix-vector multiplication over $\mathbb{F}_2$.

1: **function** MAT_VEC_MULT_SCHOOLBOOK($\mathbf{M}$, $\mathbf{v}$)
2: **for** $r \leftarrow 0$ to $(n-k-1)$ **do**
3: $\mathbf{s}_r \leftarrow 0$ ▷ Initialisation
4: **for** $c \leftarrow 0$ to $(n-1)$ **do**
5: $\mathbf{s}_r \leftarrow \mathbf{s}_r \oplus (\mathbf{M}_{[r,c]} \wedge \mathbf{v}_c)$ ▷ Multiply and add
6: **return** $\mathbf{s}$

However, for memory efficiency reasons, the schoolbook version of the matrix-vector multiplication algorithm described in Algorithm 3 is rarely used in actual

implementations. This assumes that the vector and matrix entries, which are binary, are stored as they are, each occupying a full word. To avoid occupying a full word to store only one bit, matrix row and vector entries are usually packed into words.

Packed implementation In the *packed* implementation, a vector $\mathbf{b}$ of size w is used to accumulate, by a logical XOR operation, the result of the logical AND operation between the matrix entry and the vector entry, which are both words. In this manner, the XOR and AND operations are bitsliced, and a total of w bits are processed in parallel. We refer to w as the word width. Then, this word is repeatedly shifted and XORed over itself, to perform a logical XOR operation between all its bits. Then, the least-significant bit, on which the result has been accumulated, is extracted. Eventually, this bit is packed into a syndrome word by shifting it by an amount between 0 and $w-1$ and performing a logical OR operation with the previously stored word.

We refer to this matrix-vector multiplication over $\mathbb{F}_2$ method as the *packed* method, where bits are packed into w-bit wide words. This is described in Algorithm 4, where an $(n-k) \times \frac{n}{w}$ matrix $\mathbf{M}$ is multiplied by a vector $\mathbf{v}$ of size $\frac{n}{w}$ to obtain an syndrome $\mathbf{s}$ of size $\frac{n-k}{w}$. This method is used in the reference implementation of the *Classic McEliece* cryptosystem submitted to the NIST PQC standardization process [1] with $w = 8$. Conversely, in the additional vectorized reference implementation, the value which is used is $w = 64$. The optimized implementation by Chen and Chou [14] uses $w = 32$ this time. Given the variety of the word widths used in these packed implementations, it is important to evaluate how it affects the attack success. This will be experimentally highlighted and discussed in the next sections.

Algorithm 4 Packed matrix-vector multiplication over $\mathbb{F}_2$.

function MAT_VEC_MULT_PACKED($\mathbf{M}$, $\mathbf{v}, w$)
  for $r \leftarrow 0$ to $(\frac{n-k}{w} - 1)$ **do**
    $\mathbf{s}_r \leftarrow \mathbf{0}$ ▷ Initialization
  for $r \leftarrow 0$ to $(n-k-1)$ **do**
    $\mathbf{b} \leftarrow \mathbf{0}$
    for $c \leftarrow 0$ to $(\frac{n}{w} - 1)$ **do**
      $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{M}_{[r,c]} \wedge \mathbf{v}_c)$ ▷ Multiply and add
    $i \leftarrow \frac{w}{2}$
    while $i > 0$ **do**
      $\mathbf{b} \leftarrow \mathbf{b} \oplus (\mathbf{b} >> i)$ ▷ Exclusive-OR folding
      $i \leftarrow \frac{i}{2}$
    $\mathbf{b} \leftarrow \mathbf{b} \wedge 1$ ▷ LSB extraction
    $\mathbf{s}_{r/w} \leftarrow \mathbf{s}_{r/w} \vee (\mathbf{b} << (r \bmod w))$ ▷ Bit packing
  return $\mathbf{s}$

As a side note, the schoolbook method shown in Algorithm 3 may be seen as a special case of the packed method of Algorithm 4 with $w = 1$.

3 Horizontal correlation attack on the matrix-vector multiplication

The section describes the proposed horizontal correlation attack on the *Classic McEliece* cryptosystem.

3.1 Attacker model

We place ourselves in the framework of physical attacks, where an attacker has a physical access to the device. Therefore, the attacker can measure physical quantities such as power consumption or electromagnetic radiations while the device is running. In addition, we assume that the device under attack raises a reliable trigger signal before the encapsulation. This last constraint could be relaxed, considering the regular patterns followed by the power consumption of the device while the matrix-vector multiplication over $\mathbb{F}_2$ is being performed, as shown in the next subsection.

We also assume that the attacker knows the public key $\mathbf{H}_{\text{pub}}$ and the syndrome $\mathbf{s}$, which are public, and aims at recovering the secret error vector $\mathbf{e}$ of Hamming weight t. Scenarios where the public key is generated on demand from the private key are out of scope, since the proposed attack targets the encapsulation step. Therefore, we assume that the public key is known to the attacker, and that she does not need to rely on specific techniques to obtain it. The value of t is a parameter of the cryptosystem and is assumed to be publicly known too. Even though the value of $\mathbf{s}$ is not necessary in the basic version of the attack, it is needed to carry out information-set decoding, as detailed in Subsection 3.4.

Being in the *unprofiled* attack setting, the attacker model we rely on is rather weak. In particular, we do not assume that the attacker owns a copy of the device on which he can perform a preliminary profiling step.

3.2 Side-channel trace acquisition and reshaping

After implementing Algorithm 4 in C, we performed power consumption side-channel measurements on the device running it. Experimental parameters are detailed in Section 4. An example side-channel trace of the power consumption of the microcontroller, while the matrix-vector multiplication over $\mathbb{F}_2$ is performed, is shown in Figure 1.

Since NIST explicitly requires the implementations submitted to the PQC standardization process to be constant-time, the side-channel trace is very regular and patterns can easily be spotted in it. This is visible in Figure 1, where a side-channel trace of the multiplication between a 32×64 matrix and a 64-bit vector is shown. One can easily see that a pattern is repeated 32 times, corresponding to the multiplication of the 32 matrix rows with the error vector.

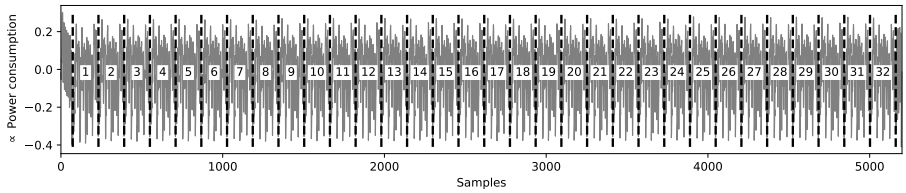


Fig. 1: Side-channel trace of the matrix-vector multiplication over $\mathbb{F}_2$ for $n = 64$ showing the $n - k = 32$ blocks.

Precise identification of this pattern's length is done by observing peaks in the auto-correlation of the side-channel trace.

Therefore, after performing the acquisition of the raw side-channel trace $\mathcal{T}_{\text{raw}}$ of length n_{samples} as depicted in Figure 1, it is reshaped into a matrix $\mathcal{T}_{\text{reshaped}}$. The process is a simpler version of the one presented in [17]. Each row of this matrix contains the side-channel leakage associated with the multiplication of one matrix row with the error vector. Therefore, there are $(n - k)$ rows in $\mathcal{T}_{\text{reshaped}}$. The number of columns in $\mathcal{T}_{\text{reshaped}}$ is approximately $\frac{n_{\text{samples}}}{n-k}$, given that a few samples at the beginning and the end of $\mathcal{T}_{\text{raw}}$ do not correspond to the matrix-vector multiplication.

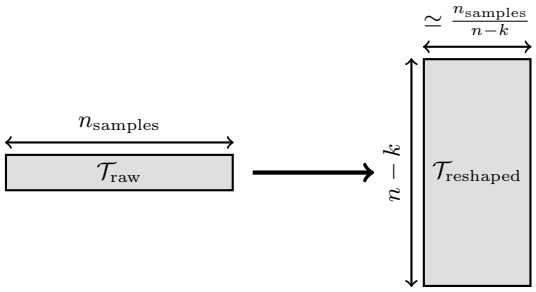


Fig. 2: Reshaping step applied on the raw trace (adapted from [17])

3.3 Horizontal Correlation Attack

The next step consists in performing a horizontal correlation attack between the columns of the parity-check matrix and sub-traces corresponding to the multiplication of one matrix row with the error vector. These sub-traces are easily identified due to the constant-time property of the matrix-vector multiplication over $\mathbb{F}_2$, as it will be shown in the Section 4. This defines a correlation matrix $\mathbf{C}$ of n rows and as many columns as the number of samples in the sub-traces. An entry of the correlation matrix $\mathbf{C}$ stores the value of the Pearson correlation

coefficient, estimated using Equation (2), between a column of the parity-check matrix and a column of the reshaped side-channel trace.

$$\rho(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}}) \cdot (\mathbf{y}_i - \bar{\mathbf{y}})}{\sqrt{\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})^2}\sqrt{\sum_{i=1}^{N}(\mathbf{y}_i - \bar{\mathbf{y}})^2}} \quad \text{with} \quad \bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i \tag{2}$$

Formally, the value of each entry of the correlation matrix $\mathbf{C}$ is given in Equation (3), where ρ stands for the Pearson correlation coefficient.

$$\mathbf{C}_{[i,j]} = \rho(\mathbf{H}_{\text{pub}_{[:,i]}}, \mathcal{T}_{\text{reshaped}_{[:,j]}}) \tag{3}$$

From there, the positions of the errors in the secret error vector are derived. This is done by keeping only the maximum value for every row of $\mathcal{C}$ and sorting them according to the maximum absolute value of the correlation coefficient. This process effectively sorts the columns of the parity-check matrix according to the probability that they are involved in the syndrome computation. Therefore, it reveals the secret error vector $\mathbf{e}$.

The intuition behind this attack path is that every bit of the error vector, no matter the width w, will have a contribution to the side-channel leakage if it meets a 1 in the associated matrix column. The asymmetry of the logical AND operation is crucial here: the side-channel leakage associated with matrix columns which are aligned with a zero in the error vector will be essentially random. Conversely, side-channel leakage associated with matrix columns which are aligned with a one in the error vector will follow the changes of values in the matrix columns.

An illustration of the proposed attack on a toy example is shown in Figure 3, with parameters $n = 8$, $k = 4$, $t = 3$ and $w = 2$.

The correlation between the columns of the reshaped side-channel trace and the matrix columns is computed to build the correlation matrix $\mathbf{C}$ in step ①. The maximum of the absolute values is computed for every column is computed in step ② and the permutation P that sorts the resulting vector is obtained in step ③. Finally, the permutation P is inverted and P^{-1} is applied to a vector made of t ones and $n - t$ zeroes to recover the secret error vector $\mathbf{e}$ in step ④.

The "n/a" entries in the $\mathbf{C}$ matrix denotes correlation values which cannot be computed because of the constant Hamming distance side-channel leakage for the last $\mathbf{b}$ computation, implying a zero variance. In reality, the side-channel leakage is always noisy and prevents such cases from happening.

3.4 Information-set decoding

Although the previous section suggested that the attack works by identifying the t matrix columns for which the absolute value of the correlation coefficient is the

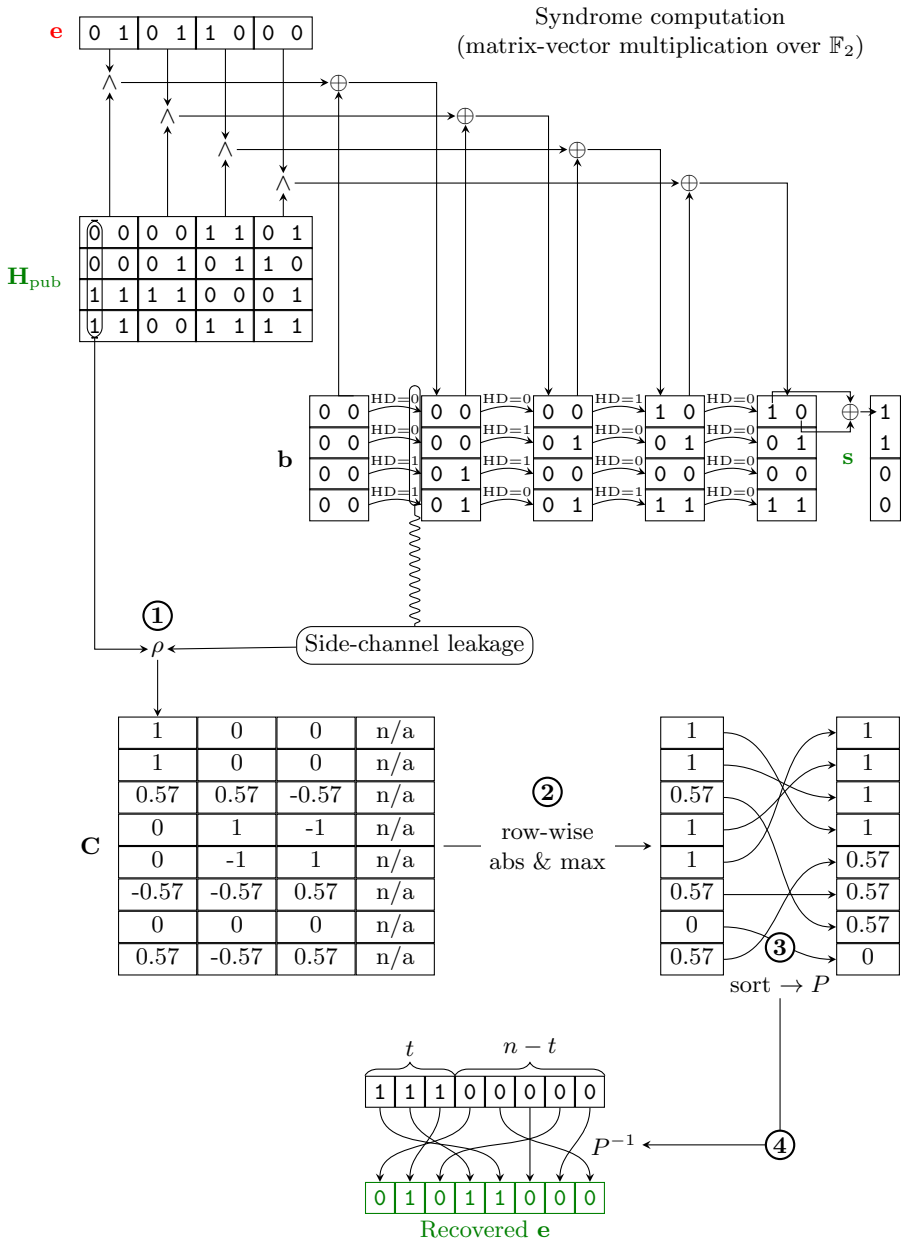


Fig. 3: Overview of the proposed horizontal attack on a toy example with parameters $n = 8$, $k = 4$, $t = 3$ and $w = 2$.

greatest, it is in fact sufficient to have those columns belong to the first $n-k$ ones, which are called the *information set*, instead of the first t columns strictly, like it

was the case for the toy example in Figure 3. First proposed by Prange [35], the information-set decoding strategy leverages linear algebra to decode more efficiently. The use of this strategy was already proposed in the context of physical attacks, to significantly improve the attack success rate [24,20,17].

Further improvements to the information-set decoding method may be used as well, to improve the success rate of this strategy [27,41,18,28] [5,29]. With these improvements, up to δ ones might be missing from the information set, as opposed to the Prange setting for which $\delta = 0$. These methods have an exponential time complexity with respect to the n parameter of the cryptosystem. Therefore, their running time quickly becomes prohibitive. However, they may still be used in a realistic setting up to $\delta = 3$. This is the value we keep for the experiments in the next section.

4 Experimental results

4.1 Simulated trace

In order to precisely control the signal-to-noise ratio (SNR) and observe its influence on the success rate of the proposed attack, we first experiment with a simulated trace. This trace is generated by considering both a Hamming weight and a Hamming distance leakage model. We did not investigate more advanced side-channel leakage simulators, like ELMO [30], since they are too device-specific. We do this at every step of Algorithm 4 where the $\mathbf{b}$ intermediate value is updated. Therefore, the Hamming weight leakage model is simulated by storing $\mathrm{HW}(\mathbf{b})$, and the Hamming distance leakage model by storing the Hamming distance between $\mathbf{b}$ and its previous value, denoted as $\mathrm{HD}(\mathbf{b}, \mathbf{b}^{-})$. The inputs $\mathbf{M}$ and $\mathbf{v}$ of the algorithm are chosen at random with $\mathrm{HW}(\mathbf{v}) = t$. Both the Hamming weight and the Hamming distance leakage are normalized, to contribute equally to the overall information leakage.

The number of ones found in the first $n-k$ positions of the error vector after sorting according to the absolute value of the correlation coefficient is shown in Figure 4. Additionally, we plot the $[t-\delta; t]$ band in green for $\delta = 3$ as justified above. If the number of ones in the first $n-k$ positions of the vector is higher than $t-\delta$, then the correct error vector can be recovered using the information-set decoding strategy outlined above. The thick black line at the bottom of the plots shows the number of ones found in the first $n-k$ positions when choosing the error vector randomly.

The experiment is run 25 times for each set (n, k, t, SNR) of parameters, and the number of ones is averaged. As expected, the higher the SNR, the easier it is to identify which columns of the parity check matrix contributed to the syndrome computation.

Two other observations can be made. First, the wider the word, the lower the success rate. This is because the contribution of a single bit to the overall correlation coefficient is less visible for large words. Therefore, the success rate of the attack is the highest for $w = 8$ and the lowest for $w = 64$. Second, the larger

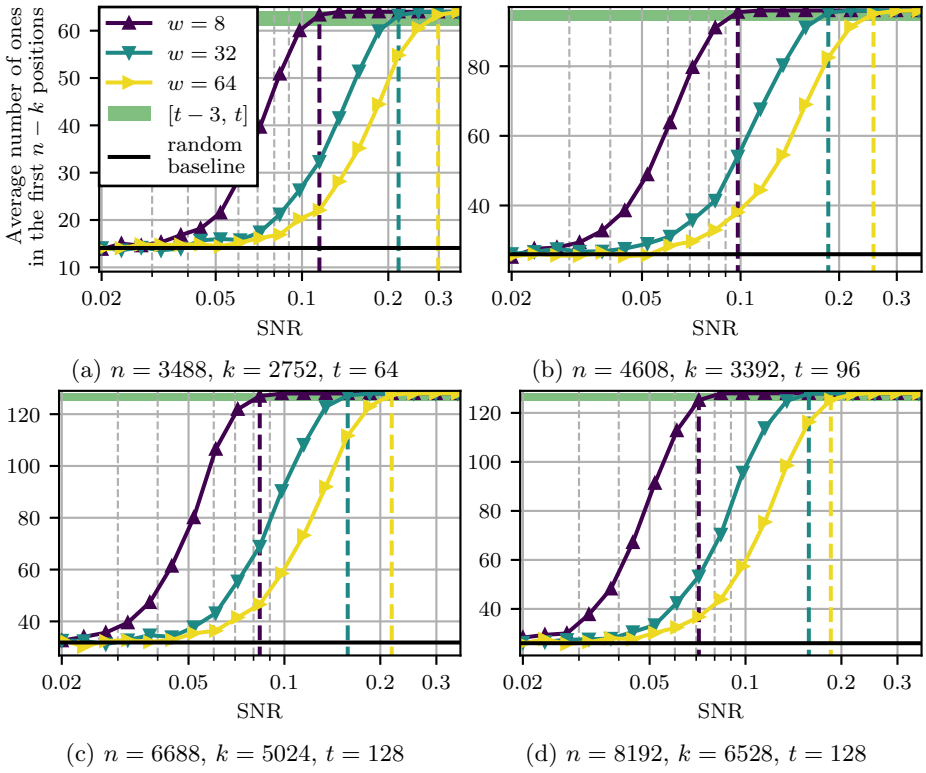


Fig. 4: Average number of ones in the first $n-k$ positions of the error vector for several SNR values for the *Classic McEliece* parameters.

the cryptographic parameters, the higher the success rate. This is explained by the fact that the error vector is used $n-k$ times in the syndrome computation. Therefore, the number of samples used in the computation of the correlation coefficient is higher for large cryptographic parameters. Therefore, the success rate of the attack is the highest for $n = 8192$ and the lowest for $n = 3488$.

4.2 Real trace

All the experiments are performed with the ChipWhisperer platform [33]. The target microcontroller embeds an ARM Cortex-M4 core, which is the embedded software target recommended by NIST for the PQC standardization process.[5] The ARM Cortex-M4 core has thirteen 32-bit registers. Therefore, one could argue that its "native" word width is $w = 32$.

Unfortunately, the particular microcontroller we used embeds only 256 kB of Flash memory. Therefore, the public key associated with even the smallest

[5] `https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/cJxMq0_90gU`

Classic McEliece parameters does not fit in, since around 331 kB would be needed to store it ($\frac{n \times m \times t}{8} = \frac{3488 \times 12 \times 64}{8} = 331\,008$).

Consequently, we must scale the cryptosystem parameters accordingly. To accommodate this constraint while keeping the parameters choice unbiased, we selected two parameters sets from the Decoding challenge webpage:[6] $(n, k, t) = (640, 512, 13)$ and $(1600, 1280, 30)$.

For each parameter set, we perform the experiments for three different word widths w: 8, 32 and 64. We extracted the relevant `syndrome` function from the NIST reference implementation. Since it uses $w = 8$ natively, we simply made two variations for $w = 32$ and $w = 64$.

Additionally, we also explore the different optimization levels available when compiling the source code for the target microcontroller: `-O0`, `-O1`, `-O2`, `-O3` and `-Os`. The first one, `-O0`, corresponds to no optimization at all. This setting is commonly used when aiming at constant-time implementations, to prevent the compiler from optimizing the loops and ruining the developer's efforts. Settings ranging from `-O1` to `-O3` correspond to various optimization efforts, `-O3` being the highest, providing the best performance. Finally, `-Os` aims at minimizing the code storage footprint in memory.

The number of clock cycles required for the multiplication of one matrix row with the error vector for the two sets of parameters are given in Table 2. As expected, both the word width and the optimization level have a strong influence on these numbers.

Word width	Optimization level				
	`-O0`	`-O1`	`-O2`	`-O3`	`-Os`
$w = 8$	3120	829	741	222	741
$w = 32$	889	229	222	226	200
$w = 64$	743	206	194	153	194

(a) $n = 640$, $k = 512$, $t = 13$

Word width	Optimization level				
	`-O0`	`-O1`	`-O2`	`-O3`	`-Os`
$w = 8$	7080	1830	1821	476	1821
$w = 32$	1939	528	522	528	470
$w = 64$	1463	456	443	443	419

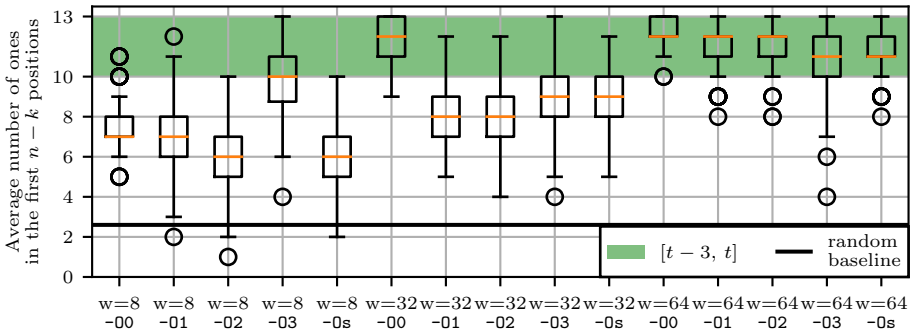
(b) $n = 1600$, $k = 1280$, $t = 30$

Table 2: Number of clock cycles for the multiplication of a matrix row by the error vector

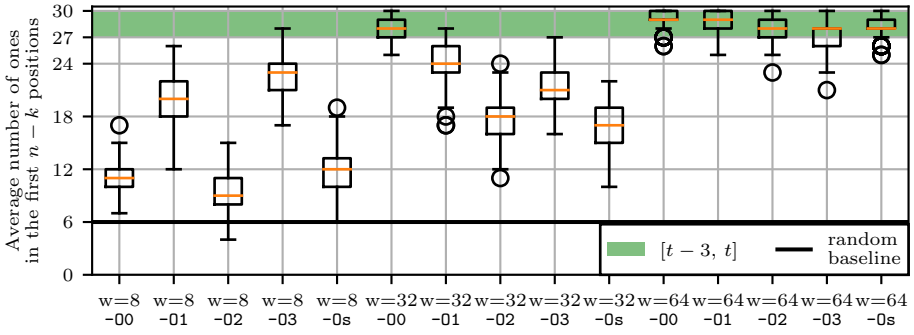
The experimental results obtained for the two parameters sets are shown in Figure 5. Each experiment was repeated 200 times to provide sufficient statistical significance for the box plots.

Several observations can be made regarding these experimental results. First of all, for a given word width w, the optimization level used for the compilation has a strong impact on the success rate of the attack, except for $w = 64$ for which this impact is lower. For example, for $w = 32$, the only successful attack is the one performed on the program which was compiled with the `-O0` option.

[6] `https://decodingchallenge.org/goppa`



(a) $n = 640$, $k = 512$, $t = 13$



(b) $n = 1600$, $k = 1280$, $t = 30$

Fig. 5: Number of ones in the first $n - k$ positions of the error vector for several word widths and compilation options for the Decoding challenge parameters.

However, this is not always the case and, contrarily to the intuition, the highest success rate is not always obtained for `-O0`. For example for $w = 8$, this is for the `-O3` optimization level that the success rate is the highest. Overall, there is no clear relation between the optimization level and the attack success rate, but it does have an impact on it.

The second observation, which again is counterintuitive and contradicts the simulation results presented in the previous subsection, is that the wider the word, the higher the success rate. For $w = 64$, no matter the optimization level, there are always more than $t - 3$ ones in the first $n - k$ positions of the error vector, therefore allowing for a full recovery using the information-set decoding strategy. This observation is hard to explain without a better understanding of the internal architecture of the processor. Some possible causes are examined in the next section.

5 Discussion on the attack success

As highlighted in the previous section, while the attack is indeed successful in several settings, its success rate seems to depend heavily on implementation choices: the word width and the optimization level. Moreover, the attack as it is described relies on a specific leakage model: the Hamming distance leakage model. The objective of this section is to take a closer look at the effect of these settings and the conditions that make the attack successful.

5.1 A closer look at the assembly

The attack success rate depends on the word width used for the implementation of the matrix-vector multiplication over $\mathbb{F}_2$ as well as the optimization level. Changing these parameters has an effect on machine code generated by the compiler. The influence of these parameters is examined below. We want to emphasize that, at the moment, we are not able to provide quantitative results about the exact role of every instruction in the observed side-channel leakage, especially with respect to the leakage model. Therefore, we restrict our analysis to a qualitative approach, highlighting the differences between the generated codes. Future works could focus on developing a complete assembly implementation of the target operation, in order to fully understand where the side-channel leakage might come from and to eliminate it.

As a preliminary step, it is important to note that the target microcontroller has 32-bit registers. Therefore, its "native" word width is 32 bits and we can expect implementations that use this width to be compiled in a more straightforward manner.

For ease of reading, the generated assembly codes are grouped together in Appendix A.

$w = 8$ When the word width is $w = 8$, then the processor must handle data which is smaller than its native register size. This is illustrated in Listings 1 to 3, where the generated codes are shown for different optimization levels. These are grouped if identical for different optimization levels.

As expected, with the lowest optimization effort `-O0`, the code is rather simple (see Listing 1). One should note that `load` and `store` instructions explicitly work with *bytes*, hence the `b` suffix (`ldrb` and `strb`). For higher optimization levels, the increment of the row and columns indices are embedded in the instruction, shortening the code but still matching precisely with the algorithm. Finally, for the highest optimization level `-O3`, the compiler determines that memory accesses can be grouped and loads words of 32 bits directly, hence the `ldr.w` variant shown in Figure 3.

The differences between the generated codes depending on the optimization level might explain why the attack success varies so much between them for $w = 8$, as shown on the left-hand side of Figure 5b.

w = 32 When the word width is $w = 32$, then the processor handles data which is of its native register size. Therefore, the generated code is more consistent across optimization levels, as shown in Listings 4 to 6. This might explain why the success rate of the attack for $w = 32$ seems more consistent, as shown in the center of Figure 5b. However, there is no apparent reason why the attack only succeeds for the `-O0` optimization level and not for the others.

We chose to not include the generated code for the `-O3` optimization level. Indeed, it is different depending on the chosen n value. For $n = 640$, the inner `for` loop, found at line 6 in Algorithm 4, is fully unrolled. This is not the case for the larger value $n = 1600$, for which the generated code is very similar to the one obtained with optimization levels `-O2` and `-Os`, shown in Listing 6.

w = 64 When the word width is $w = 64$, then the processor must handle data which is larger than its native register size. As such, when loading data, either a load double `ldrd` instruction or two distinct load `ldr.w` instructions are used. We denote those as `step 1/2` in Listings 7 to 10. Some other instructions might be inserted between the two steps to better fill up the pipeline. Again, we do not include the code generated for the `-O3` optimization level because of inconsistent loop unrolling depending on the value of n. Besides this difference, the code is the same as for $w = 32$. Therefore, similarly to the conclusion we drew for $w = 32$, it is hard to find a reason why the attack is almost always successful for $w = 64$.

5.2 Leakage assessment

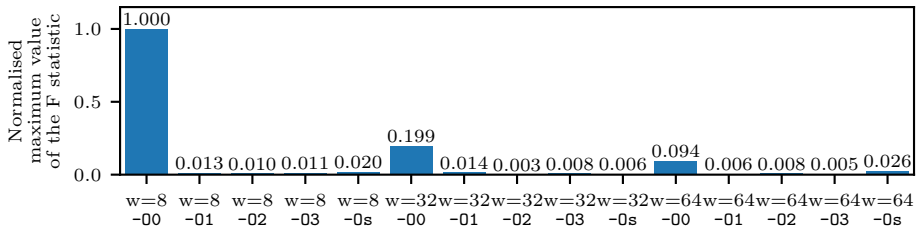
To try to better understand the differences of success rate depending on the word width and optimization levels, we conducted a leakage assessment on the side-channel traces. We used a rather common ANOVA (ANalysis Of VAriance) F-test, also referred to as NICV [8], which is a ratio between the inter-class and intra-class variances, where classes are intermediate values of the syndrome computation after application of a leakage model. We consider both the Hamming weight and Hamming distance leakage models in this analysis.

We perform the F-test on side-channel traces obtained with $n = 1600$ and $k = 1280$. Since we consider intermediate values of $\mathbf{b}$, this leaves us with a population of size of $\frac{(n-k)\times n}{w}$. For the largest value of $w = 64$ this is a population of 8000, which is statistically relevant considering the number of classes at hand. Instead of plotting the values of the F statistic for all samples, we keep only the maximum for a given set of parameters. In addition, we normalize it since what we are interested in is a comparison between values for a given leakage model.

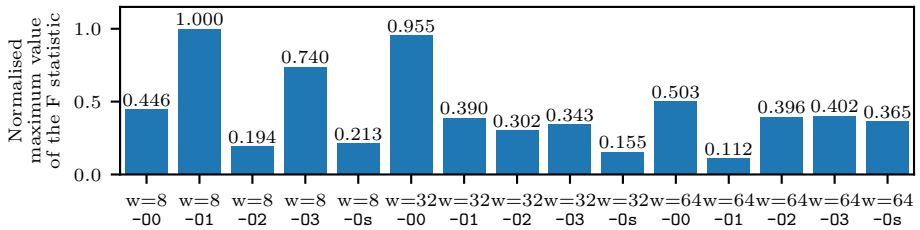
Results are shown in Figure 6. For the Hamming weight leakage model, in Figure 6a, the highest F statistic values are systematically obtained for the lowest level of optimization `-O0`. Intuitively, this can be explained because the code is less condensed than for higher levels of optimization, allowing to better observe the Hamming weight values through the power consumption. The F statistic values are much lower for higher optimization levels. Nevertheless, even though the F statistic is very high for $w = 8$ and the `-O0` optimization level, the attack

does not succeed in this case, as shown in Figure 5. Therefore, it confirms the intuition that the attack is not possible with the Hamming weight leakage model.

For the Hamming distance leakage model, in Figure 6b, which is the one being exploited by the proposed attack, the variations observed for the different parameters almost follow the success rates from Figure 5. This is a confirmation that the leakage model being exploited experimentally is really the Hamming distance leakage model.



(a) Hamming weight leakage model



(b) Hamming distance leakage model

Fig. 6: Maximum value of the F statistic for several word widths and compilation options and two leakage models.

5.3 Dependency on the Hamming distance leakage model

The attack path is better explained using a Hamming distance leakage model, directly illustrating the transitions from 0 to 1 or from 1 to 0 when updating the **b** value, as shown in line 7 of Algorithm 4. A Hamming weight leakage model would not allow to follow the proposed attack path, since, for a Hamming distance of 1, the Hamming weight might either increase or decrease, which makes it useless when computing the correlation.

Nevertheless, when implementing and compiling the line 7 of Algorithm 4, it is unavoidable that several machine code instructions are generated. In particular, the logical AND between the matrix entry and the error vector entry is computed *first*, before being accumulated on **b** by a logical XOR *later*. As a consequence, a Hamming weight leakage model on this intermediate value, storing the logical AND between the matrix entry and the error vector entry, could

be leveraged for the attack as well. This is formalized in Equation (4) after the definition of the Hamming distance.

$$\mathrm{HD}(\mathbf{b}, \mathbf{b}^{\text{-}}) = \mathrm{HD}(\mathbf{b}^{\text{-}} \oplus \mathbf{H}_{\mathrm{pub}_{[i,j]}} \wedge \mathbf{e}_j, \mathbf{b}^{\text{-}}) = \mathrm{HW}(\mathbf{H}_{\mathrm{pub}_{[i,j]}} \wedge \mathbf{e}_j) \tag{4}$$

Another related issue with the Hamming weight or Hamming distance leakage models is the so-called "double-cancellation" phenomenon [17,19]. It happens when two bits of the intermediate value $\mathbf{b}$ are flipped in opposite directions. While this is visible on the Hamming distance, this is hidden when considering only the Hamming weights. However, it requires two ones to be in the same word of width w in $\mathbf{e}$, which is quite rare and not an issue for the proposed correlation-based attack.

5.4 Hardware implementations

While the experimental results provided show the attack success on a microcontroller, there is a strong possibility that the same attack path could be exploited for hardware implementations too. Indeed, the method of splitting the public-key matrix $\mathbf{H}_{\mathrm{pub}}$ and the error vector $\mathbf{e}$ in words of width w was also chosen in the FPGA implementation of *Classic McEliece* [15]. Consequently, the Hamming distance between intermediate values stored in the flip-flops of the digital design could be retrieved by side-channel analysis, and the same attack would apply. The Hamming distance model is usually very applicable when performing power or electromagnetic analysis of FPGA and ASIC implementations [34,4]. As a side note, a mixture of a Hamming weight and Hamming distance leakage was already exploited in a similar attack on the QC-MDPC McEliece cryptosystem in [13].

6 Conclusion

This article introduces the first unprofiled attack on the post-quantum cryptosystem *Classic McEliece*. Exploiting the constant-time property of the matrix-vector multiplication over $\mathbb{F}_2$ used in the encapsulation, the proposed horizontal correlation attack recovers the secret error vector $\mathbf{e}$, and therefore the KEM shared secret key, from a single side-channel trace.

We highlighted how implementation choices, namely the word width used for the bitsliced operations and the optimization level passed to the compiler, have a strong impact on the attack success rate. Using a leakage assessment methodology and disassembling the generated code, we tried to explain the success rate variations.

As next steps, it is worth investigating the feasibility of the attack on a different architecture, like RISC-V, or on a hardware target, like an FPGA. Another interesting aspect could be to better exploit the Hamming weight leakage, since it is very strong for the `-O0` optimization level. Combining both the Hamming weight and the Hamming distance leakages could also lead to more powerful attacks.

Acknowledgements

This work received funding from the France 2030 program, managed by the French National Research Agency under grant agreement No. ANR-22-PETQ-0008 PQ-TLS.

References

1. Albrecht, M.R., Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Paterson, K.G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C.J., Tomlinson, M., Wang, W.: Classic McEliece: conservative code-based cryptography: cryptosystem specification. Tech. rep., National Institute of Standards and Technology (2022)
2. Avanzi, R., Hoerder, S., Page, D., Tunstall, M.: Side-channel attacks on the McEliece and Niederreiter public-key cryptosystems. Journal of Cryptograhic Engineering **1**(4), 271–281 (2011)
3. Aydin, F., Aysu, A., Tiwari, M., Gerstlauer, A., Orshansky, M.: Horizontal side-channel vulnerabilities of post-quantum key exchange and encapsulation protocols. ACM Transactions on Embedded Computing Systems **20**(6), 110:1–110:22 (2021)
4. Aysu, A., Tobah, Y., Tiwari, M., Gerstlauer, A., Orshansky, M.: Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In: IEEE International Symposium on Hardware Oriented Security and Trust. pp. 81–88. IEEE Computer Society, Washington, DC, USA (Apr 2018)
5. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 7237, pp. 520–536. Springer, Cambridge, UK (Apr 2012)
6. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.A.: On the inherent intractability of certain coding problems (corresp.). IEEE Transactions on Information Theory **24**(3), 384–386 (1978)
7. Beullens, W., D'Anvers, J., Hülsing, A., Lange, T., Panny, L., Saint Guilhem, C., Smart, N.: Post-quantum cryptography : current state and quantum mitigation. Publications Office of the European Union (2022). `https://doi.org/doi/10.2824/92307`
8. Bhasin, S., Danger, J., Guilley, S., Najm, Z.: NICV: normalized inter-class variance for detection of side-channel leakage. IACR Cryptology ePrint Archive p. 717 (2013)
9. Biasi, F.P., Barreto, P.S.L.M., Misoczki, R., Ruggiero, W.V.: Scaling efficient code-based cryptosystems for embedded platforms. Journal of Cryptographic Engineering **4**(2), 123–134 (2014)
10. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J. (eds.) International Workshop on Cryptographic Hardware and Embedded Systems. Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer, Cambridge, MA, USA (Aug 2004)
11. Cayrel, P.L., Colombier, B., Dragoi, V.F., Menu, A., Bossuet, L.: Message-recovery laser fault injection attack on the classic mceliece cryptosystem. In: Canteaut, A., Standaert, F.X. (eds.) Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science,

vol. 12697, pp. 438–467. Springer, Zagreb, Croatia (10 2021). https://doi.org/10.1007/978-3-030-77886-6_15
12. Cayrel, P.L., Dusart, P.: McEliece/Niederreiter PKC: Sensitivity to fault injection. In: International Conference on Future Information Technology. Busan, South Korea (May 2010)
13. Chen, C., Eisenbarth, T., von Maurich, I., Steinwandt, R.: Horizontal and vertical side channel analysis of a McEliece cryptosystem. IEEE Transactions on Information Forensics and Secuity. **11**(6), 1093–1105 (2016)
14. Chen, M., Chou, T.: Classic McEliece on the ARM Cortex-M4. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(3), 125–148 (2021)
15. Chen, P., Chou, T., Deshpande, S., Lahr, N., Niederhagen, R., Szefer, J., Wang, W.: Complete and improved FPGA implementation of Classic McEliece. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(3), 71–113 (2022)
16. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal correlation analysis on exponentiation. In: Soriano, M., Qing, S., López, J. (eds.) International Conference on Information and Communications Security. Lecture Notes in Computer Science, vol. 6476, pp. 46–61. Springer, Barcelona, Spain (Dec 2010)
17. Colombier, B., Dragoi, V.F., Cayrel, P.L., Grosso, V.: Profiled side-channel attack on cryptosystems based on the binary syndrome decoding problem. IEEE Transactions on Information Forensics and Security **17**, 3407–3420 (2022). https://doi.org/10.1109/TIFS.2022.3198277
18. Dumer, I.: On minimum distance decoding of linear codes. In: Joint Soviet-Swedish International Workshop on Information Theory. pp. 50–52. Moscow, Russia (Jan 1991)
19. Grosso, V., Cayrel, P.L., Colombier, B., Dragoi, V.F.: Punctured syndrome decoding problem - efficient side-channel attacks against classic mceliece. In: Kavun, E.B., Pehl, M. (eds.) International Workshop on Constructive Side-Channel Analysis and Secure Design. Lecture Notes in Computer Science, vol. 13979, pp. 170–192. Springer, Munich, Germany (4 2023). https://doi.org/10.1007/978-3-031-29497-6_9
20. Guo, Q., Johansson, A., Johansson, T.: A key-recovery side-channel attack on Classic McEliece implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 800–827 (2022)
21. Heyse, S.: Low-reiter: Niederreiter encryption scheme for embedded microcontrollers. In: Sendrier, N. (ed.) International Workshop on Post-Quantum Cryptography. Lecture Notes in Computer Science, vol. 6061, pp. 165–181. Springer, Darmstadt, Germany (May 2010)
22. Heyse, S., Moradi, A., Paar, C.: Practical power analysis attacks on software implementations of mceliece. In: Sendrier, N. (ed.) Third International Workshop on Post-Quantum Cryptography. Lecture Notes in Computer Science, vol. 6061, pp. 108–125. Springer, Darmstadt, Germany (May 2010)
23. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Annual International Cryptology Conference. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer, Santa Barbara, CA, USA (Aug 1999)
24. Lahr, N., Niederhagen, R., Petri, R., Samardjiska, S.: Side channel information set decoding using iterative chunking - plaintext recovery from the "Classic McEliece" hardware reference implementation. In: Moriai, S., Wang, H. (eds.) Annual International Conference on the Theory and Application of Cryptology and Information

Security. Lecture Notes in Computer Science, vol. 12491, pp. 881–910. Springer, Daejeon, South Korea (Dec 2020)
25. LaRacuente, N., Smith, K.N., Imany, P., Silverman, K.L., Chong, F.T.: Modeling short-range microwave networks to scale superconducting quantum computation (2023)
26. Larsen, M.V., Guo, X., Breum, C.R., Neergaard-Nielsen, J.S., Andersen, U.L.: Deterministic multi-mode gates on a scalable photonic quantum computing platform. Nature Physics **17**(9), 1018–1023 (2021)
27. Lee, P.J., Brickell, E.F.: An observation on the security of McEliece's public-key cryptosystem. In: Günther, C.G. (ed.) Workshop on the Theory and Application of of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 330, pp. 275–280. Springer, Davos, Switzerland (May 1988)
28. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) International Conference on the Theory and Application of Cryptology and Information Security. Lecture Notes in Computer Science, vol. 7073, pp. 107–124. Springer, Seoul, South Korea (Dec 2011)
29. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 9056, pp. 203–228. Springer, Sofia, Bulgaria (Apr 2015)
30. McCann, D., Oswald, E., Whitnall, C.: Towards practical tools for side channel aware software engineering: 'grey box' modelling for instruction leakages. In: Kirda, E., Ristenpart, T. (eds.) USENIX Security Symposium. pp. 199–216. USENIX Association, Vancouver, BC, Canada (Aug 2017)
31. Molter, H.G., Stöttinger, M., Shoufan, A., Strenzke, F.: A simple power analysis attack on a mceliece cryptoprocessor. Journal of Cryptographic Engineering **1**(1), 29–36 (2011). https://doi.org/10.1007/s13389-011-0001-3, https://doi.org/10.1007/s13389-011-0001-3
32. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. Problems of Control and Information Theory **15**(2), 159–166 (1986)
33. O'Flynn, C., Chen, Z.: Chipwhisperer: An open-source platform for hardware embedded security research. In: Prouff, E. (ed.) International Workshop on Constructive Side-Channel Analysis and Secure Design. Lecture Notes in Computer Science, vol. 8622, pp. 243–260. Springer, Paris, France (Apr 2014)
34. Peeters, E., Standaert, F., Quisquater, J.: Power and electromagnetic analysis: Improved model, consequences and comparisons. Integration **40**(1), 52–60 (2007)
35. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
36. Preskill, J.: Quantum computing in the nisq era and beyond. Quantum **2**, 79 (2018)
37. Ravi, P., Chattopadhyay, A., D'Anvers, J.P., Baksi, A.: Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Survey and new results. Cryptology ePrint Archive, Paper 2022/737 (2022), https://eprint.iacr.org/2022/737, https://eprint.iacr.org/2022/737
38. Roth, J., Karatsiolis, E.G., Krämer, J.: Classic McEliece implementation with low memory footprint. In: Liardet, P., Mentens, N. (eds.) International Conference on Smart Card Research and Advanced Applications. Lecture Notes in Computer Science, vol. 12609, pp. 34–49. Springer, Virtual Event (Nov 2020)

39. Saarinen, M.J.O.: Wip: Applicability of iso standard side-channel leakage tests to nist post-quantum cryptography. In: 2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 69–72 (2022). https://doi.org/10.1109/HOST54066.2022.9839849
40. Shoufan, A., Strenzke, F., Molter, H.G., Stöttinger, M.: A timing attack against patterson algorithm in the McEliece PKC. In: Lee, D.H., Hong, S. (eds.) International Conference on Information, Security and Cryptology. Lecture Notes in Computer Science, vol. 5984, pp. 161–175. Springer, Seoul, Korea (Dec 2009)
41. Stern, J.: A method for finding codewords of small weight. In: Cohen, G.D., Wolfmann, J. (eds.) International Colloquium on Coding Theory and Applications. Lecture Notes in Computer Science, vol. 388, pp. 106–113. Springer, Toulon, France (Nov 1988)
42. Strenzke, F.: Solutions for the storage problem of McEliece public and private keys on memory-constrained platforms. In: Gollmann, D., Freiling, F.C. (eds.) International Conference on Information Security. Lecture Notes in Computer Science, vol. 7483, pp. 120–135. Springer, Passau, Germany (Sep 2012)
43. Sundaresan, N., Lauer, I., Pritchett, E., Magesan, E., Jurcevic, P., Gambetta, J.M.: Reducing unitary and spectator errors in cross resonance with optimized rotary echoes. PRX Quantum **1**(2) (dec 2020). https://doi.org/10.1103/prxquantum.1.020318, https://doi.org/10.1103%2Fprxquantum.1.020318
44. Takeda, S., Furusawa, A.: Toward large-scale fault-tolerant universal photonic quantum computing. APL Photonics **4**(6), 060902 (2019)
45. Walter, C.D.: Sliding windows succumbs to big mac attack. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) International Workshop on Cryptographic Hardware and Embedded Systems. Lecture Notes in Computer Science, vol. 2162, pp. 286–299. Springer, Paris, France (May 2001)

A Assembly codes for different optimization levels and word widths

All the assembly codes shown below were obtained with the `arm-none-eabi-gcc` compiler, version 9.2.1 20191025. Annotations are by the authors and follow the notations used in the article.

Listing 1 $w = 8$, optimization level `-O0`

```
ldrb r2, [r3, #0]   // Load byte e_j
...
ldrb r3, [r3, #0]   // Load byte H_pub[i,j]
ands r3, r2         // H_pub[i,j] ∧ e_j
uxtb r2, r3         // Unsigned extend byte
ldrb r3, [r7, #31]  // Load byte b
eors r3, r2         // b = b ⊕ H_pub[i,j] ∧ e_j
strb r3, [r7, #31]  // Store byte b
```

Listing 2 $w = 8$, optimization levels `-O1`, `-O2` and `-Os`

```
ldrb.w r3, [r2, #1]!  // Load byte H_pub[i,j]
ldrb.w lr, [ip, #1]!  // Load byte e_j
and.w  r3, r3, lr     // H_pub[i,j] ∧ e_j
eors   r1, r3         // b = b ⊕ H_pub[i,j] ∧ e_j
```

Listing 3 $w = 8$, optimization level `-O3`

```
ldr.w r4, [r2], #4  // Load 4 bytes H_pub[i,j], H_pub[i,j+1], H_pub[i,j+2] and H_pub[i,j+3]
ldr.w r3, [r0], #4  // Load 4 bytes e_j, e_j+1, e_j+2 and e_j+3
...
and.w r3, r3, r4    // H_pub[i,j] ∧ e_j, H_pub[i,j+1] ∧ e_j+1, H_pub[i,j+2] ∧ e_j+2
                    // and H_pub[i,j+3] ∧ e_j+3
eor.w r1, r1, r3    // b_j = b_j ⊕ H_pub[i,j] ∧ e_j, b_j+1 = b_j+1 ⊕ H_pub[i,j+1] ∧ e_j+1
                    // b_j+2 = b_j+2 ⊕ H_pub[i,j+2] ∧ e_j+2 and b_j+3 = b_j+3 ⊕ H_pub[i,j+3] ∧ e_j+3
```

Listing 4 $w = 32$, optimization level `-O0`

```
ldr  r2, [r3, #0]   // Load e_j
...
ldr  r3, [r3, #0]   // Load H_pub[i,j]
ands r3, r2         // H_pub[i,j] ∧ e_j
ldr  r2, [r7, #36]  // Load b
eors r3, r2         // b = b ⊕ H_pub[i,j] ∧ e_j
str  r3, [r7, #36]  // Store b
```

Listing 5 $w = 32$, optimization level `-O1`

```
ldr.w r3, [r2, #4]!  // Load H_pub[i,j];
ldr.w r5, [r7, #4]!  // Load e_j;
ands  r3, r5         // H_pub[i,j] ∧ e_j
eors  r1, r3         // b = b ⊕ H_pub[i,j] ∧ e_j
```

Listing 6 $w = 32$, optimization levels `-O2` and `-Os`

```
ldr.w r4, [r2], #4  // Load H_pub[i,j];
ldr.w r3, [r0], #4  // Load e_j;
...
and.w r3, r3, r4    // H_pub[i,j] ∧ e_j
eor.w r1, r1, r3    // b = b ⊕ H_pub[i,j] ∧ e_j
```

Listing 7 $w = 64$, optimization level `-O0`

```
ldrd  r0, r1, [r3]        // Load H_pub[i,j] in 2 registers
...
ldrd  r2, r3, [r3]        // Load e_j in 2 registers
and.w r5, r0, r2          // H_pub[i,j] ∧ e_j (step 1)
and.w r6, r1, r3          // H_pub[i,j] ∧ e_j (step 2)
ldrd  r2, r3, [r7, #112]  // Load b in 2 registers
eor.w r1, r2, r5          // b = b ⊕ H_pub[i,j] ∧ e_j (step 1)
str   r1, [r7, #56]       // Store b (step 1)
eors  r3, r6              // b = b ⊕ H_pub[i,j] ∧ e_j (step 2)
str   r3, [r7, #60]       // Store b (step 2)
```

Listing 8 $w = 64$, optimization level `-O1`

```
ldr.w r1, [r3, #8]!  // Load H_pub[i,j] (step 1)
ldr   r2, [r3, #4]   // Load e_j (step 1)
ldr.w r7, [r4, #8]!  // Load H_pub[i,j] (step 2)
ands  r1, r7         // H_pub[i,j] ∧ e_j (step 1)
ldr   r7, [r4, #4]   // Load e_j (step 2)
ands  r2, r7         // H_pub[i,j] ∧ e_j (step 2)
eor.w ip, r1, ip     // b = b ⊕ H_pub[i,j] ∧ e_j (step 1)
eors  r0, r2         // b = b ⊕ H_pub[i,j] ∧ e_j (step 2)
```

Listing 9 $w = 64$, optimization level `-O2`

```
ldr.w r1, [r3, #8]!  // Load H_pub[i,j] (step 1)
ldr.w r6, [r4, #8]!  // Load e_j (step 1)
ldr   r2, [r3, #4]   // Load H_pub[i,j] (step 2)
ands  r1, r6         // H_pub[i,j] ∧ e_j (step 1)
ldr   r6, [r4, #4]   // Load e_j (step 2)
...
and.w r2, r2, r6     // H_pub[i,j] ∧ e_j (step 2)
eor.w r5, r5, r1     // b = b ⊕ H_pub[i,j] ∧ e_j (step 1)
eor.w r0, r0, r2     // b = b ⊕ H_pub[i,j] ∧ e_j (step 2)
```

Listing 10 $w = 64$, optimization level `-Os`

```
ldr.w fp, [r4, #8]!  // Load H_pub[i,j] (step 1)
ldrd  sl, r1, [r8]   // Load e_j in 2 registers
ldr   r7, [r4, #4]   // Load H_pub[i,j] (step 2)
and.w sl, sl, fp     // H_pub[i,j] ∧ e_j (step 1)
ands  r1, r7         // H_pub[i,j] ∧ e_j (step 2)
...
eor.w r0, sl, r0     // b = b ⊕ H_pub[i,j] ∧ e_j (step 1)
eor.w r3, r3, r1     // b = b ⊕ H_pub[i,j] ∧ e_j (step 2)
```