Message-recovery Profiled Side-channel Attack on the Classic McEliece Cryptosystem

Brice Colombier[1], Vlad-Florin Drăgoi[2,3], Pierre-Louis Cayrel[4] and Vincent Grosso[4]

[1] Univ Grenoble Alpes, CNRS, Grenoble INP, TIMA, 38000 Grenoble, France
brice.colombier@grenoble-inp.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
[4] Univ Lyon, UJM-Saint-Etienne, CNRS, Laboratoire Hubert Curien UMR 5516, F-42023, Saint-Etienne, France
{pierre.louis.cayrel;vincent.grosso}@univ-st-etienne.fr

Abstract. The NIST standardization process for post-quantum cryptography has been drawing the attention of researchers to the submitted candidates. One direction of research consists in implementing those candidates on embedded systems and that exposes them to physical attacks in return. The *Classic McEliece* cryptosystem, which is among the four finalists of round 3 in the Key Encapsulation Mechanism category, was recently targeted by a laser fault injection attack leading to message recovery. Regrettably, the attack setting is very restrictive. Indeed, it does not tolerate errors in the faulty syndrome. Moreover, it depends on the very strong attacker model of laser fault injection, and is not applicable to optimised implementations of the algorithm that make optimal usage of the machine words capacity. In this article, we propose a change of attack angle and perform a message-recovery attack that relies on side-channel information only. We improve on the previously published work in several key aspects. First, we show that side-channel information is sufficient to obtain a faulty syndrome in $\mathbb{N}$, as required by the attack. This is done by leveraging classic machine learning techniques that recover the Hamming weight information very accurately. Second, we put forward a computationally-efficient method, based on a simple dot product, to recover the message from the, possibly noisy, syndrome in $\mathbb{N}$. We show that this new method, which additionally leverages existing information-set decoding algorithms from coding theory, is very robust to noise. Finally, we present a countermeasure against the proposed attack.

Keywords: Post-quantum Cryptography · Classic McEliece · Side-channel Attack

1 Introduction

The post-quantum cryptography standardisation process, initiated and supported by the NIST, has almost reached its conclusion, as third round candidates were announced on July 2020[1]. One of the submissions, the Classic McEliece cryptosystem [ABC+20], is based on the theory of error-correcting codes. It instantiates the Niederreiter cryptosystem [Nie86] with binary Goppa codes, which is the dual of the McEliece cryptosystem [McE78]. The parameters of the cryptosystem are set with respect to the complexity of the best information-set decoding attack strategy [MMT11, BJMM12, MO15, BM18, EMZ21], which is the best known general attack path against code-based cryptosystems. As the scheme started to gain scientific confidence, sustained efforts were directed towards the practical side, i.e., implementations [Hey10, HG13, vMHG16, WSN18, BSNK19, DFA+20, RKK20, CC21, KRF+21], and physical attacks, both side-channel [LNPS20] and fault injection attacks [DK20, XIU+21, CCD+21].

[1] https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions

Message-recovery attacks A message-recovery attack that uses side-channel leakage during the decryption process was proposed in [LNPS20]. The central idea of this article is to add random columns of the parity matrix to the syndrome. The modified syndrome is sent to a decoding oracle, on which side-channel analysis is carried out, to detect if the weight of the input message (error) is higher than expected. If the weight of the input message increases, that means the message bits of the corresponding columns are 0 in most positions. In the other case, if the weight decreases, the message bits of the corresponding columns are 1 in most positions. Information-set decoding is used to reduce the number of calls to the oracle.

In [CCD+21] the authors recover the message using laser-fault injection during the encryption process. By performing a bit-set laser fault, the matrix-vector multiplication over $\mathbb{F}_2$ is changed into a matrix-vector multiplications over $\mathbb{N}$. By solving a modified version of the syndrome decoding problem, they manage to recover the message within minutes.

Key-recovery attacks Some attacks aim at recovering the secret key instead. In [DK20], a fault injection attack framework is presented, but it requires quite a large number of faults, injected in a precise manner, and its success rate is at most 50 %. No practical attack was mounted in this work.

A generic attack on the Fujisaki-Okamoto transformation was proposed in [XIU+21] and performed on multiple candidates submitted to the NIST standardisation process. This generic attack relies on the ability to skip an instruction during the decapsulation process. Interestingly, this attack relies on a single fault, making it very practical, as demonstrated experimentally in [XIU+21]. However, it has not been applied to the Classic McEliece cryptosystem.

Modified versions of the syndrome decoding problem The idea of augmenting the syndrome decoding problem (SDP) where some extra information, mainly gained by means of physical attacks, has been studied in [CCD+21, HPR+21]. While in [HPR+21] the overall cost of the attacks remain unfeasible practically, in [CCD+21], all the parameters set of the Classic McEliece were attacked. The Integer Syndrome Decoding ($\mathbb{N}-\mathsf{SDP}$), introduced in [DCC+20, CCD+21], is a similar problem to the SDP. The difference is that the matrix-vector multiplication is performed over $\mathbb{N}$ instead of $\mathbb{F}_2$. Solutions to this problem can be found by using Integer Linear Programming algorithms such as the simplex [DCC+20] or interior-points methods [CCD+21]. The latter is very computationally efficient, and can solve the $\mathbb{N}-\mathsf{SDP}$ using only a small proportion of the parity-check rows. The authors show by simulations that the empirical complexity is $\mathcal{O}(n^3)$. However, there are several drawbacks to this algorithm: its exact complexity remains unknown and it does not tolerate any errors in the syndrome. This is a major problem in the context of physical attacks, where perfect reproducibility during the fault injection process is rarely obtained. Hence, a different approach has to be considered in order to overcome these issues.

It turns out that $\mathbb{N}-\mathsf{SDP}$ was already considered in the literature, first by Dorfman [Dor43], and is known as the group testing problem or quantitative group testing [LCPR19, GHKL19, FL20]. Some of the algorithms from [FL20] are in this article, with some modifications, imposed by the context of side-channel analysis. Further analysis is also needed to fit these algorithms in the framework of information-set decoding methods.

Contributions In this article, a hybrid message recovery attack against the Classic McEliece cryptosystem is performed. By hybrid, we refer to physical attacks and theoretical results combined together in order to retrieve the initial message given the ciphertext, also called the syndrome, which is a binary vector. The side-channel attack is performed on the matrix-vector multiplication during the encryption process. By means of random forests, the value of the syndrome in $\mathbb{N}$ is recovered and is used later as input for the $\mathbb{N}-\mathsf{SDP}$. Finally, we show that there is an algorithm that can retrieve the initial message from the syndrome in $\mathbb{F}_2$ in polynomial time in the code length. Let us detail a bit more each step and contribution.

First of all, we introduce a novel method to obtain the syndrome in $\mathbb{N}$ instead of $\mathbb{F}_2$ as required by the attack framework presented in [CCD+21]. While the initial attack relied on

an expensive and restrictive laser fault injection setup, the method we propose here is based on side-channel analysis only. Using random forests, this method recovers the syndrome in $\mathbb{N}$ from Hamming weight information during the syndrome computation. Another significant difference, compared to laser fault injection attacks, is that no modification of the ciphertext is performed during the side-channel attack. Hence, at the end of this stage, an adversary has two vectors of information: the real ciphertext, which is a binary syndrome, and an approximate version the same syndrome in $\mathbb{N}$, obtained by side-channel analysis. These steps are presented in Section 3, after a review of code-based cryptography and the Classic McEliece cryptosystem is done in Section 2, focusing in particular on hardware implementations.

Second, we develop another way to recover the positions of the errors in the error vector, that is, the message, from the syndrome in $\mathbb{N}$, as detailed in Section 4. The initial attack in [CCD+21] made use of integer linear programming solvers for this task. Although these solvers are quite efficient, they do not tolerate any error in the syndrome in $\mathbb{N}$. We introduce an alternative method, based on a simple dot product operation, that acts as a powerful distinguisher even in the presence of errors. More precisely, using the values of the syndrome in $\mathbb{N}$ and the public key, i.e., parity-check matrix $\boldsymbol{H}$, we deduce a permutation on the columns of $\boldsymbol{H}$. The permuted parity-check matrix is then set into standard form. Our permutation is able to move almost all the non-zero positions of the error-vector on the support of the identity in the standard form of the permuted parity-check matrix. It mainly acts as an almost "perfect" permutation for an Information Set Decoding (ISD) algorithm. Hence, the algorithm we propose has to main steps. First, we use an approximation of the syndrome in $\mathbb{N}$ to find a "good" permutation. Second, we use it in addition to the syndrome in $\mathbb{F}_2$, that is, the exact ciphertext, in an ISD variant to retrieve the initial message, that is, the binary error vector, solution to the syndrome decoding problem. We evaluate the resistance of this method to errors in the syndrome in $\mathbb{N}$ and show that it improves both computationally and with respect to the resistance to errors when compared to the integer linear programming techniques used in [CCD+21]. Section 5 presents the experimental results.

Finally, we pinpoint the weakness of the Classic McEliece cryptosystem that makes it vulnerable to the proposed attack. We then suggest a countermeasure, introduced in Section 6 that makes the proposed attack much harder.

2 Code-based cryptosystems

2.1 Coding theory

Notations The following conventions and notations are used. A finite field is denoted by $\mathbb{F}$, and the ring of integers by $\mathbb{Z}$. We denote $\mathbb{N}_n^* = \{1, \dots, n\}$ and $\mathbb{Z}_{-n,n} = \{-n, \dots, 0, \dots, n\}$. For $p \in [0, 1]$ and $n \in \mathbb{N}^*$ we denote the Bernoulli distribution by $\mathcal{B}er(p)$ and the Binomial distribution by $\mathcal{B}(n, p)$.

Matrices and vectors are written in bold capital, respectively small letters, e.g., a vector of length n is $\boldsymbol{c} = (c_1, \dots, c_n)$ and a $k \times n$ matrix is $\boldsymbol{H} = (h_{j,i})_{(j,i) \in \mathbb{N}_k^* \times \mathbb{N}_n^*}$. We will use the following notation for sub-matrices. $\boldsymbol{H}_{[i,]}$ represents the i^{th} row of $\boldsymbol{H}$ and $\boldsymbol{H}_{[,i]}$ the i^{th} column of $\boldsymbol{H}$. Binary vectors and matrices will be packed into blocks of 8-bit sub-vectors and sub-matrices. For example, we will write $\boldsymbol{c} = (\boldsymbol{c}_1, \dots, \boldsymbol{c}_{n/8})$, where $\boldsymbol{c}_{i+1} = (c_{8i+1}, \dots, c_{8i+8})$ for vectors, and $\boldsymbol{H}_{[i,j]}$ is an 8-bit vector defined as $\boldsymbol{H}_{[i,j+1]} = (h_{i,8j+1}, \dots, h_{i,8j+8})$ for matrices. The set of all $k \times n$ matrices over $\mathbb{F}$ is $\mathbb{F}^{k \times n}$. The support of a vector, that is, the positions of its non-zero coordinates, is defined as $\mathsf{Supp}(\boldsymbol{e}) = \{i \in \mathbb{N}_n^* \mid e_i \neq 0\}$. The concatenation of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is written as $\boldsymbol{a} \parallel \boldsymbol{b}$. The Hamming weight of a binary vector, that is, the number of non-zero coordinates, is written as $\mathrm{HW}(\boldsymbol{e})$. The Hamming distance between $\boldsymbol{a}$ and $\boldsymbol{b}$ is written as $\mathrm{HD}(\boldsymbol{a}, \boldsymbol{b})$.

Error correcting codes Let n and k be two positive integers such that $k \leq n$. An $[n, k]$ linear error correcting code, or simply a linear code, is a sub-vector space of dimension k

of the vector space $\mathbb{F}^n$. It is defined either by its generator matrix $\boldsymbol{G} \in \mathbb{F}^{k\times n}$, which is a basis for the code, or by its parity-check matrix $\boldsymbol{H} \in \mathbb{F}^{(n-k)\times n}$, which is a basis for the dual code, where $\boldsymbol{G}\boldsymbol{H}^t = \boldsymbol{0}$. The minimum distance, or the Hamming distance of a code $\mathcal{C}$, is the minimum of all $\mathrm{HW}(v)$ for $\boldsymbol{v} \in \mathcal{C}, \boldsymbol{v} \neq \boldsymbol{0}$. A linear code with Hamming distance d can correct $t = \lfloor (d-1)/2 \rfloor$ errors. Retrieving the error pattern is a fundamental concept in coding theory. A possible way of redefining it is by means of the well-known syndrome decoding problem.

Definition 1 (Syndrome decoding problem SDP)**.**
Inputs: $\boldsymbol{H} \in \mathbb{F}^{(n-k)\times n}$, $\boldsymbol{s} \in \mathbb{F}^{n-k}$, $t \in \mathbb{N}^*$.
Output: $\boldsymbol{e} \in \mathbb{F}^n$ with $\mathrm{HW}(\boldsymbol{e}) = t$, such that $\boldsymbol{H}\boldsymbol{e} = \boldsymbol{s}$.

Here, as for the majority of the code-based encryption schemes, we will restrict to binary codes, i.e., $\mathbb{F} = \mathbb{F}_2$. The decisional version of the SDP, also known as the coset weight problem, belongs to the class of NP-complete problems [BMvT78]. Hence, all the existing algorithms for solving SDP are exponential in the code parameters [TS16]. Moreover, for $t = \mathcal{O}(n)$ and k/n constant, the work factor of state-of-the-art algorithms for SDP tends towards $2^{-t\log(1-k/n)(1+o(1))}$, when n goes to infinity. A non-exhaustive list of algorithms for SDP includes [Pra62, Ste88, LB88, Leo88, Dum89, CC98, FS09, MMT11, BJMM12, MO15, BM18].

2.2 The Niederreiter cryptosystem

The *Classic McEliece* cryptosystem [ABC+20] is based on the Niederreiter cryptosystem [Nie86]. The key generation, encryption and decryption functions of the Niederreiter cryptosystem are given in Algorithms 1, 2 and 3 respectively.

In this article, we focus on the encryption step, in particular on the syndrome computation (line 3 in Algorithm 2). Its implementation, as a matrix-vector multiplication, is detailed below.

Algorithm 1 Niederreiter key generation

1: **function** KeyGen(n, k, t)
2: $\mathcal{C}$ an $[n, k]$ code that corrects t errors
3: A parity-check of $\mathcal{C}$: $\boldsymbol{H}$
4: An $n \times n$ permutation matrix $\boldsymbol{P}$
5: An $(n-k) \times (n-k)$ invertible matrix $\boldsymbol{S}$
6: Compute $\boldsymbol{H}_{\mathrm{pub}} = \boldsymbol{S}\boldsymbol{H}\boldsymbol{P}$
7: $\mathsf{pk} = (\boldsymbol{H}_{\mathrm{pub}}, t)$
8: $\mathsf{sk} = (\boldsymbol{S}, \boldsymbol{H}, \boldsymbol{P})$
9: **return** $(\mathsf{pk}, \mathsf{sk})$

Algorithm 2 Niederreiter encryption

1: **function** Encrypt($\boldsymbol{m}$, pk)
2: Encode $\boldsymbol{m} \to \boldsymbol{e}$ with $\mathrm{HW}(\boldsymbol{e}) = t$
3: Compute $\boldsymbol{s} = \boldsymbol{H}_{\mathrm{pub}}\boldsymbol{e}$
4: **return** $\boldsymbol{s}$

Algorithm 3 Niederreiter decryption

1: **function** Decrypt($\boldsymbol{s}$, sk)
2: Compute $\boldsymbol{e}' = \mathrm{Decode}(\boldsymbol{S}^{-1}\boldsymbol{s}, \boldsymbol{H})$
3: Compute $\boldsymbol{m}$ from $\boldsymbol{P}^{-1}\boldsymbol{e}'$
4: **return** $\boldsymbol{m}$

The sets of (n, k, t) parameters suggested in [ABC+20] are given in Table 1. Instantiating the Niederreiter scheme with binary Goppa codes will be considered, as proposed by the authors of the submission [ABC+20].

2.3 Hardware implementations

Before the NIST started the post-quantum cryptography standardisation process, a number of hardware implementations of the Niederreiter cryptosystem had been proposed. We

Table 1: *Classic McEliece* parameters

Parameters set	`348864`	`460896`	`6688128`	`8192128`
n	3488	4608	6688	8192
k	2720	3360	5024	6528
t	64	96	128	128

refer to both microcontroller and FPGA implementations as hardware implementations here. Interestingly, the McEliece cryptosystem was the first to be implemented in hardware [EGHP09]. The Niederreiter cryptosystem has then been implemented on FPGA [Hey10, HG13, WSN18] for high performance and on microcontroller [BCS13, vMHG16, Cho18] for embedded systems.

Later on, as the *Classic McEliece* cryptosystem entered the NIST standardisation process, a number of hardware implementations were proposed. First, in order to benchmark the proposals, several hardware/software co-design and high-level synthesis methods were explored to speed up the implementation process and obtain rough figures for comparison purposes [BSNK19, DFA+20, KRF+21]. Later on, more specific microcontroller implementations of the candidates were released [KRSS19]. However, the *Classic McEliece* cryptosystem was not included in this comparison since, as already noted in [CC21], the public keys were too large to fit in the memory of the targeted platform. Nevertheless, a few works showed that such an implementation is in fact feasible, by heavily optimising numerous steps of the key generation, encryption and decryption processes [RKK20, CC21].

***Classic McEliece* NIST PQC submission** The *Classic McEliece* submission to the NIST PQC standardisation process comes with C source code, as required. In this source code, the syndrome computation is implemented in a *packed* fashion, as shown in Algorithm 4, meaning that the bits of the matrix and the error-vector in $\mathbb{F}_2$ are packed into bytes to better exploit the capacity of the machine words.

The main step of the syndrome computation is shown on line 7 in Algorithm 4, where an intermediate value b, which is a byte, is repeatedly exclusive-ORed with the bitwise AND between one byte from the matrix row and one byte from the error vector. In [CCD+21], only the *schoolbook* version of the matrix-vector multiplication algorithm is attacked, where bits are stored individually in the machine words. In the attack we propose here, we target the packed version, although the schoolbook version could be attacked in the same way.

Algorithm 4 Packed matrix-vector multiplication

```
1:  function MAT_VEC_MULT_PACKED(H, e)
2:    for r ← 0 to ((n − k)/8 − 1) do
3:      s_r = 0                                          ▷ Initialisation
4:    for r ← 0 to (n − k − 1) do
5:      b = 0
6:      for c ← 0 to (n/8 − 1) do
7:        b ^= H_[r,c] & e_c                  ▷ Multiplication and addition
8:      b ^= b ≫ 4                                       ▷
9:      b ^= b ≫ 2                                       ▷ Exclusive-OR folding
10:     b ^= b ≫ 1                                       ▷
11:     b &= 1                                           ▷ LSB extraction
12:     s_⌊r/8⌋ |= b ≪ (r mod 8)                         ▷ Bit packing
13:   return s
```

2.4 Message-recovery attack on the *Classic McEliece* cryptosystem

In this section, we outline the attack proposed in [CCD+21], since the attack we propose follows the same framework.

The first step consists in obtaining a syndrome in $\mathbb{N}$ instead of $\mathbb{F}_2$. To this end, in [CCD+21], an exclusive-OR instruction is corrupted into an ADD instruction by laser fault injection. Here, we propose to use side-channel analysis only to accomplish this task. Laser fault injection has several drawbacks compared to side-channel analysis for this attack setting.

First, the number of faults to inject exhibits quadratic growth with respect to n, since $n.(n-k) = \mathcal{O}(n^2)$ faults are required. Even though the constant-time algorithm implementation make the fault injection feasible in practice, the number of faults gets very large for realistic values of n. For instance, in the *Classic McEliece* cryptosystem, since $3488 \leq n \leq 8192$, more than one million faults must be injected.

Second, while the laser fault injection attack applies to the schoolbook version of the matrix-vector multiplication algorithm, in which each machine word stores only one bit of information, it does not adapt easily to the packed version of this algorithm shown in Algorithm 4.

Finally, it involves a lot of resources and a very strong attacker model. Indeed, the attacker must own an expensive laser fault injection setup and have access to the backside of the chip to perform laser fault injection through the bulk. Even though this is a standard attacker model for laser fault injection, it calls for simplification and a less restrictive way to carry out the attack.

The second step of the attack consists in recovering the message from the syndrome in $\mathbb{N}$. To this end, in [CCD+21], a computationally-efficient integer linear programming solver is used. Here, we propose to perform a dot product between the faulty syndrome in $\mathbb{N}$ and the columns of the parity-check matrix to recover the positions of the errors in the error vector, as detailed in Section 4.

Compared with the attack proposed in [CCD+21], the one we describe here is much more practical, since it is based on side-channel analysis only, it is faster, and less sensitive to errors. A side-by-side comparison of the two attacks is provided in Table 2.

Table 2: Comparison of [CCD+21] and the proposed attack

Attack step	**Attack in [CCD+21]**	**Proposed attack**
Method to obtain a syndrome in $\mathbb{N}$	instruction corruption with laser fault injection	side-channel attack
Derivation of the exact value of the syndrome in $\mathbb{N}$	yes	no
Multiplication method that can be attacked	schoolbook only	schoolbook and packed
Detectable alterations of the ciphertext (syndrome)	yes	no
Message recovery from the syndrome in $\mathbb{N}$	integer linear programming	dot-product and information-set decoding
Practicality	expensive setup and restrictive attacker model	very applicable

3 Side-channel recovery of the syndrome in $\mathbb{N}$

The first step of the attack on the *Classic McEliece* cryptosystem proposed in [CCD+21] consists in obtaining a syndrome in $\mathbb{N}$ instead of $\mathbb{F}_2$. In this section, we propose an alternative

way to obtain the syndrome in $\mathbb{N}$, that does not rely on fault injection, but on side-channel measurements only.

3.1 Side-channel measurements

We perform side-channel measurements during the syndrome computation, that is, when the syndrome $\boldsymbol{s}$ is computed as the matrix-vector multiplication $\boldsymbol{He} = \boldsymbol{s}$. Being in the profiled attack setting, we record a training set for which the inputs of the matrix-vector multiplication algorithm are chosen. Actually, as detailed below, a single trace is sufficient to form the training set. For this trace, both the matrix $\boldsymbol{H}$ and the error vector $\boldsymbol{e}$ are random, and $\mathrm{HW}(\boldsymbol{e}) = t$. Although a single trace is used, samples diversity is ensured by the fact that the matrix $\boldsymbol{H}$ is very large and random and that the trace is preprocessed before being fed to the classifier. The trace is composed of n_{samples} and is stored as a vector $\boldsymbol{t}_{\text{raw}}$. We additionally record a second trace, $\boldsymbol{t}_{\text{test}}$, that is used as a test set when training the classifier. For both traces, we also store the Hamming weights of the intermediate value $\boldsymbol{b}$ used in the syndrome computation (see line 7 in Algorithm 4). They will be used as labels for the classifier.

3.2 Side-channel traces preprocessing

Considering the dimensions of the $\boldsymbol{H}$ matrix involved in the syndrome computation, the number of samples n_{samples} is very large, and grows quadratically with respect to n. Thus the trace cannot be fed directly to a classifier and must be preprocessed first. This three-step procedure is described below.

3.2.1 First reshaping : matrix-row level

The first preprocessing step consists in reshaping the vector $\boldsymbol{t}_{\text{raw}}$ into a matrix, with each row corresponding to the multiplication of one row of $\boldsymbol{H}$ with the error vector $\boldsymbol{e}$. This is easily done since the matrix-vector multiplication is a very regular operation, in which patterns can be identified by visual inspection.

What we obtain is a matrix of traces $\boldsymbol{T}_{\text{row-wise}}$ of $(n-k)$ rows and approximately $\frac{n_{\text{samples}}}{n-k}$ columns. The number of columns is not fixed, since it depends on the number of non-informative samples that were recorded at the beginning and the end of the trace, before the matrix-vector multiplication started and after it was finished. The number of columns is actually the number of samples corresponding to the multiplication of one row of the matrix $\boldsymbol{H}$ by the error vector $\boldsymbol{e}$.

3.2.2 Second reshaping : matrix-element level

The second preprocessing step is similar to the first and consists in reshaping again the matrix of traces, so that each row now corresponds to the multiplication of one element of $\boldsymbol{H}$ with one element of the error vector $\boldsymbol{e}$. This is easily done since the vector-vector multiplication is a very regular operation, in which patterns can be identified easily.

What we obtain is a new matrix of traces $\boldsymbol{T}_{\text{element}}$ of $(n-k).\frac{n}{8}$ rows and approximately $\frac{n_{\text{samples}}}{\frac{n}{8}.(n-k)}$ columns. Again, the number of columns is not fixed, since it depends on the number of non-informative samples that were recorded at the beginning and the end of each preprocessed trace, before the vector-vector multiplication started and after it was finished. In fact, these extra samples correspond to the exclusive-OR folding operation, the LSB extraction and the bit-level storage of the syndrome (see lines 8 to 12 in Algorithm 4).

3.2.3 Linear Discriminant Analysis

After those two steps, each row of the matrix corresponds to the multiplication of one element of the matrix $\boldsymbol{H}$ with one element of the error vector $\boldsymbol{e}$. We then apply Linear Discriminant

Analysis to reduce the dimensions of the rows of the matrix of traces. This will make them easier to handle by the classifier.

Linear Discriminant Analysis is a powerful alternative to Principal Component Analysis for dimensionality reduction. As stated in [SA08], while Principal Component Analysis aims at maximising the variance of the mean traces, Linear Discriminant Analysis aims at maximising the ratio between the inter-class variance and the intra-class variance. As a dimensionality-reduction technique, Linear Discriminant Analysis maps the input data into a space of dimension $n_{\text{classes}} - 1$. This can be understood intuitively by considering that a *one*-dimensional space is sufficient to allow for a linear separation between *two* classes. However, to linearly separate *three* classes, a *two*-dimensional space is required. In our case, since we aim at recovering the Hamming weight of bytes, there are nine classes and the $\boldsymbol{T}_{\text{LDA}}$ matrix has eight columns.

3.2.4 Summary

Figure 1 summarises the preprocessing steps which are applied to the raw side-channel measurements, showing the changes in the vector/matrix dimensions at every step.

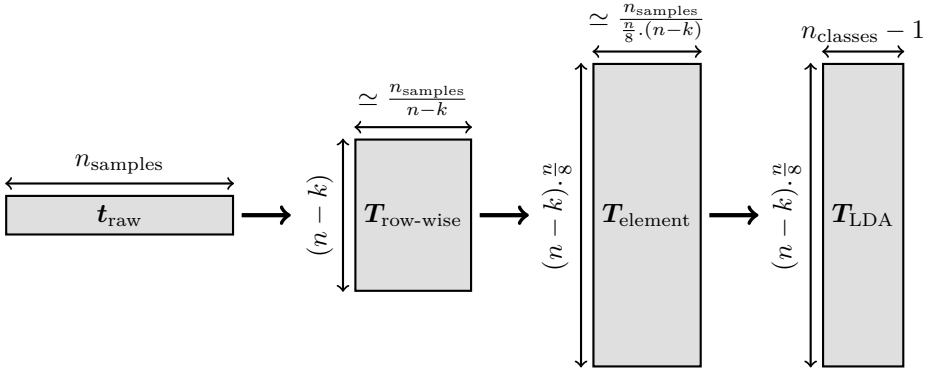


Figure 1: Summary of the preprocessing steps applied to the raw traces

These preprocessing steps actually have two very interesting advantages when considering the complexity of the training process. These two advantages directly stem from the fact that the traces are reshaped *twice*, which is made possible by the constant-time property of the implementation of the algorithm.

The first advantage is that, in the final matrix $\boldsymbol{T}_{\text{LDA}}$, each row corresponds to the multiplication of one entry of $\boldsymbol{H}$ with one entry of $\boldsymbol{e}$. Therefore, even though the value of n changes, only *one* classifier must be trained. This is very interesting from a training complexity point of view, since as n grows, the number of classifiers to train remains *constant*, and equal to one.

A second advantage, which directly derives from the one above, is that the number of training samples extracted from a single trace in $\boldsymbol{t}_{\text{raw}}$ grows *quadratically* with respect to n. Indeed, as shown in Figure 1, the number of rows in $\boldsymbol{T}_{\text{LDA}}$ is $(n-k).\frac{n}{8} = \mathcal{O}(n^2)$. For example, if $n = 8192$ and $k = 6528$, then a *single* side-channel trace provides $\frac{n}{8} \times k = 1024 \times 6528 \approx 6.7 \times 10^6$ training samples. Consequently, provided that an attacker can record a sufficiently large number of samples, then a single side-channel trace is sufficient to build the training set.

3.3 Hamming weight recovery with a random forest

After the side-channel trace has been preprocessed following the steps described above, the matrix and labels are fed to a classifier which is trained to recover the Hamming weight of the intermediate value $\boldsymbol{b}$ used in the syndrome computation. To this end, we selected the random forest algorithm, which is more lightweight than deep learning neural networks. It has been used previously for side-channel analysis with good results [HGG20].

A random forest is a classifier that belongs to the category of ensemble learning. As such, it is composed of an ensemble of decision trees, and the outcome of their individual decisions is combined by majority voting. We refer the reader to [Bre01] for more details. As classically done, the classifier is trained with the samples and labels from the training set and its accuracy is evaluated with the samples and labels from the test set $\boldsymbol{t}_{\text{test}}$.

Improving the classification accuracy Although the classification accuracy of the default random forest was already rather high, we further improved it by considering the rows of $\boldsymbol{T}_{\text{LDA}}$ that come before and after the one being classified (see Equation (1)). Intuitively, this originates from the fact that the Hamming weight of the intermediate value $\boldsymbol{b}$ in fact does not vary much. Indeed, since the error vector $\boldsymbol{e}$ has a low Hamming weight, then there is a high probability that consecutive values of the intermediate value $\boldsymbol{b}$ have the same Hamming weight. Therefore, by feeding the classifier with augmented data obtained by concatenating previous and consecutive rows of $\boldsymbol{T}_{\text{LDA}}$, we can greatly improve the classification accuracy. This can also be seen as having a window of width $(2.\Delta + 1)$ sliding vertically over the matrix $\boldsymbol{T}_{\text{LDA}}$.

$$\boldsymbol{T}_{\text{LDA ext.}[i,]} = \boldsymbol{T}_{\text{LDA}[i-\Delta,]} \parallel \boldsymbol{T}_{\text{LDA}[i-\Delta+1,]} \parallel \ldots \parallel \boldsymbol{T}_{\text{LDA}[i,]} \parallel \ldots \parallel \boldsymbol{T}_{\text{LDA}[i+\Delta-1,]} \parallel \boldsymbol{T}_{\text{LDA}[i+\Delta,]} \quad (1)$$

Edge cases for $i < \Delta$ or $i > (n - \Delta)$ are handled by duplicating the closest matrix row if needed. Therefore, the vector fed to the classifier $\boldsymbol{T}_{\text{LDA ext.}}[i]$ always has $(2.\Delta + 1).(n_{\text{classes}} - 1)$ components.

The Δ value is empirically selected. Experiments presented in Section 5.3 show that $\Delta = 3$ is a good choice and leads to a very high classification accuracy, while keeping the computational and memory complexity of the dimensionality reduction step reasonable.

3.4 Derivation of the approximate syndrome in $\mathbb{N}$

Once the consecutive Hamming weights of the intermediate value are recovered, an approximate value for the syndrome entries in $\mathbb{N}$ may be derived. Actually, instead of the Hamming weights, having the Hamming distances would allow to derive the *exact* value for the syndrome entries, as shown by Equation (2). Indeed, the Hamming distance between two consecutive values of $\boldsymbol{b}$ is exactly the number of 1s found in the bitwise AND between the byte from the matrix row and the byte from the error vector (see line 7 in Algorithm 4). Computing the value of the syndrome entry in $\mathbb{N}$ is equivalent to counting those ones, which in turn is equivalent to summing the Hamming distances between consecutive values of $\boldsymbol{b}$. Considering that the intermediate value is an 8-bit vector written as $\boldsymbol{b}$, and having $\boldsymbol{b}_{j,0} = \boldsymbol{0}$ for all $j \in \{1, \ldots, n-k\}$ we deduce the j^{th} syndrome entry s_j from Equation (2).

$$1 \leq j \leq (n-k) \quad s_j = \sum_{i=1}^{\frac{n}{8}} \text{HD}(\boldsymbol{b}_{j,i}, \boldsymbol{b}_{j,i-1}) \quad (2)$$

Unfortunately, the leakage model we observed in the experiments is a Hamming weight leakage model, not a Hamming distance leakage model. This claim is supported by signal-to-noise ratio computations presented in Section 5. Therefore, we propose another formula to derive the value of the syndrome entry in $\mathbb{N}$ from the Hamming weights, given in Equation (3).

$$1 \leq j \leq (n-k) \quad s_j = \sum_{i=1}^{\frac{n}{8}} \left| \text{HW}(\boldsymbol{b}_{j,i}) - \text{HW}(\boldsymbol{b}_{j,i-1}) \right| \quad \text{if } \text{HD}(\boldsymbol{b}_{j,i}, \boldsymbol{b}_{j,i-1}) \leq 1 \quad (3)$$

Lemma 1. *Let* $\boldsymbol{1}_\ell = (1, \ldots, 1) \in \mathbb{F}_2^\ell$ *and* $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_2^\ell$ *with* $\text{HW}(\boldsymbol{a}) \geq \text{HW}(\boldsymbol{b})$. *Let* $\overline{\boldsymbol{a}} = \boldsymbol{1}_\ell - \boldsymbol{a}$ *be the bitwise complement of* $\boldsymbol{a}$. *Let* $\boldsymbol{a}\&\boldsymbol{b} = (a_1 b_1, \ldots, a_\ell b_\ell)$ *be the element-wise product of the vectors* $\boldsymbol{a}$ *and* $\boldsymbol{b}$. *Then we have:*

$$\text{HD}(\boldsymbol{a}, \boldsymbol{b}) = \text{HW}(\boldsymbol{a}) - \text{HW}(\boldsymbol{b}) + 2.\text{HW}(\overline{\boldsymbol{a}}\&\boldsymbol{b}). \quad (4)$$

In particular, if $\mathsf{Supp}(\boldsymbol{b}) \subseteq \mathsf{Supp}(\boldsymbol{a})$ *we have*

$$\mathrm{HD}(\boldsymbol{a}, \boldsymbol{b}) = \mathrm{HW}(\boldsymbol{a}) - \mathrm{HW}(\boldsymbol{b}). \tag{5}$$

Notice that a particular case where the aforementioned condition $\mathsf{Supp}(\boldsymbol{b}) \subseteq \mathsf{Supp}(\boldsymbol{a})$ is valid is when $\mathrm{HD}(\boldsymbol{a}, \boldsymbol{b}) \leq 1$, as specified in Equation (3).

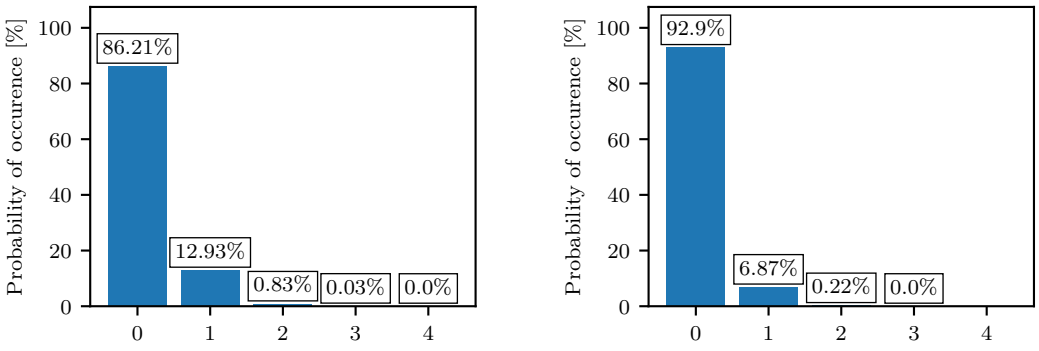
In the general case, considering random bytes, the Hamming distance between two bytes is, most of the time, *not* equal to the absolute value of the difference of their Hamming weights. This can be checked exhaustively: the two values are different more than 80 % of the time, and the condition of Equation (3) mostly does not hold.

However, we are not in this general case here, since the error vector is of low Hamming weight (see the *Classic McEliece* parameters in Table 1). For example, for the lowest value of $n = 3488$, the error vector has a Hamming weight of 64 only. For the highest value of $n = 8192$, the error vector has a Hamming weight of 128.

Therefore, the bitwise AND between the byte from the matrix (of Hamming weight equal to 4 on average) and the byte from the error vector (of low Hamming weight) is also of low Hamming weight. As a consequence, since this byte is exclusive-ORed with the previous value of $\boldsymbol{b}$ (see line 7 in Algorithm 4), the Hamming distance between two consecutive values of $\boldsymbol{b}_{i,j}$ will be small and less than one most of the time, satisfying the aforementioned condition.

As an illustrative example, let us consider the first set of parameters for *Classic McEliece* where $n = 3488$ and $t = 64$. Figure 2a shows the empirical distribution of the Hamming weight of the bytes found in a vector of length $n = 3488$ and Hamming weight $t = 64$. This in accordance with the fact that, since $t = 64$, then there are at least $\frac{n}{8} - t$ bytes with a Hamming weight of zero, or $\frac{\frac{n}{8}-t}{\frac{n}{8}} = \frac{436-64}{436} = \frac{372}{436} = 85.32\,\%$.

We can see in Figure 2a that, indeed, the vast majority of these bytes have a Hamming weight of zero or one, with less than one percent having a Hamming weight of two or more. This is the distribution of the Hamming weight of the bytes in the error vector.



(a) Empirical distribution of the Hamming weight of the bytes found in a vector of length $n = 3488$ and of Hamming weight $t = 64$.

(b) Empirical distribution of the Hamming weight of the bitwise AND between the bytes found in a vector of length $n = 3488$ and of Hamming weight $t = 64$ and a random byte.

Figure 2: Distributions of the Hamming weight of the intermediate values involved in the syndrome computation.

Figure 2b illustrates the empirical distribution of the Hamming weight of the bitwise AND between the bytes found in a vector of length $n = 3488$ and of Hamming weight $t = 64$ and a random byte. We can see that more than 99.75 % of these bytes have a Hamming weight of zero or one. This is the distribution of the bitwise AND between the bytes in the matrix row and the bytes in the error vector. Since this value is exclusive-ORed with $\boldsymbol{b}$ in the syndrome computation, then the distribution shown in Figure 2b is also the distribution of the Hamming

distances between consecutive values of $\boldsymbol{b}_{i,j}$. Therefore, in our case, the Hamming distance between two consecutive values of $\boldsymbol{b}_{i,j}$ is, most of the time, *equal* to the absolute value of the difference of their Hamming weights.

While, as shown in Figure 2b, Equation (3) holds more than 99.75 % of the time, it is invalid if the Hamming distance is equal to two or more. In this case, information about the Hamming weights is not equivalent to information from the Hamming distances. However, since this occurs very rarely, it has a very low impact on the recovery of the syndrome in $\mathbb{N}$.

It is important to note that the derivation of the syndrome in $\mathbb{N}$ from the Hamming weights *always underestimates* it. This is stated in Equation (4), which can also be checked exhaustively. Note that this is true only if the classifier is perfect, that is, if Hamming weights are perfectly recovered. This can never be exactly the case, as detailed experimentally in Section 5.

Extra parity-correction step After estimating the syndrome in $\mathbb{N}$ thanks to the Hamming weight information and Equation (3), an additional parity correction step can be followed. Indeed, the syndrome in $\mathbb{F}_2$ is known: this is the ciphertext. This gives us information about the parity of the syndrome in $\mathbb{N}$. Since the recovered syndrome in $\mathbb{N}$ is always underestimated, one can add the parity value to the estimated value.

For example, if we estimate the value of a syndrome entry in $\mathbb{N}$ to be 14, but the associated syndrome entry in $\mathbb{F}_2$ is 1, then we know that the value 14 is wrong, since it is even but the syndrome entry in $\mathbb{F}_2$ is odd. The estimated syndrome entry in $\mathbb{N}$ is then changed to 15.

What we eventually obtain is an estimation of the syndrome in $\mathbb{N}$. More precisely $\boldsymbol{s} \bmod 2 \in \mathbb{F}_2^{n-k}$ is the correct syndrome but the entries of $\boldsymbol{s}$ might be incorrect (but with the correct parity). Although this estimation of $\boldsymbol{s}$ might be incorrect, the next section shows how to exploit it to recover the message.

4 Message recovery algorithms

The second step of the attack on the *Classic McEliece* cryptosystem proposed in [CCD+21] consisted in using linear programming algorithms to solve the $\mathbb{N} - \mathsf{SDP}$. Those algorithms have several drawbacks. First, they are not resistant to errors during the acquisition in the first attack step: a single incorrect entry in the syndrome in $\mathbb{N}$ leads to an incorrect result. Thus the syndrome in $\mathbb{N}$ must be perfectly correct. Second, the complexity of those algorithms was not analysed in [CCD+21]. In this section, we propose an alternative way to solve the $\mathbb{N} - \mathsf{SDP}$ with a dedicated algorithm and show that it resists errors in the syndrome. We also provide a clear analysis of the complexity of this algorithm.

We start by recalling the definition of $\mathbb{N} - \mathsf{SDP}$ in Definition 2. The only difference between the $\mathbb{N} - \mathsf{SDP}$ and the SDP, shown in Definition 1, is that the syndrome is in $\mathbb{N}$ instead of $\mathbb{F}_2$.

Definition 2 (Integer syndrome decoding problem $\mathbb{N} - \mathsf{SDP}$)**.**
Inputs: $\boldsymbol{H} \in \{0,1\}^{(n-k)\times n}$, $\boldsymbol{s} \in \mathbb{N}^{n-k}$, $t \in \mathbb{N}^*$.
Output: $\boldsymbol{e} \in \{0,1\}^n$ with $\mathrm{HW}(\boldsymbol{e}) = t$, such that $\boldsymbol{H}\boldsymbol{e} = \boldsymbol{s}$.

4.1 Identification of a *good* permutation

The main idea behind the algorithms proposed here is to use techniques such as those in [FL20] in order to find a *good* initial permutation for the information-set decoder.

4.1.1 Score function

Compared with classical approaches where permutations are exhaustively enumerated, we exploit the extra-information coming from the syndrome in $\mathbb{N}$, and determine a better permutation. To do so, we assign a score for each column of $\boldsymbol{H}$, computed as a scalar product between the column in question and the syndrome in $\mathbb{N}$. The intuition behind using the scalar

product is that, since the columns are high-dimensional, then there is a high probability that they "point" in very different directions in the high-dimensional space. Since they are summed to get the syndrome in $\mathbb{N}$, the scalar product should find which columns point in the same direction as the syndrome in $\mathbb{N}$.

In order to increase the correlation between columns in the support of the error vector and the syndrome vector we will use extra information by means of the complement, as stated in Lemma 2. Next, sorting the columns of $\boldsymbol{H}$ with respect to their corresponding score will determine a column permutation on $\boldsymbol{H}$. With this permutation at hand we use an efficient variant of information-set decoding to find the correct error vector. We shall begin by providing a useful fact related to any solution of the $\mathbb{N}-\mathsf{SDP}$.

Lemma 2. *Let $\boldsymbol{e}$ be a valid solution for the $\mathbb{N}-\mathsf{SDP}$ with input $\boldsymbol{H}, \boldsymbol{s}, t$. Then $\boldsymbol{e}$ is a valid solution for an equivalent $\mathbb{N}-\mathsf{SDP}$ with input $\overline{\boldsymbol{H}}, \overline{\boldsymbol{s}}, t$ where $\overline{h_{j,i}} = 1 - h_{j,i}, \forall (j,i) \in \mathbb{N}^*_{n-k} \times \mathbb{N}^*_n$ and $\overline{s_j} = t - s_j, \forall j \in \mathbb{N}^*_{n-k}$.*

What this lemma implies is that we can double the number of equations by considering both the complements $\overline{\boldsymbol{H}}$ and $\overline{\boldsymbol{s}}$ in addition to $\boldsymbol{H}$ and $\boldsymbol{s}$. We now define the score function ψ in Definition 3.

Definition 3 (Score function)**.**
Let $\boldsymbol{H} \in \{0,1\}^{(n-k)\times n}, \boldsymbol{s} \in \mathbb{N}^{n-k}$ and $t \in \mathbb{N}^*$. Define the following score function:

$$\forall i \in \mathbb{N}^*_n \quad \psi_i(\boldsymbol{s}) = \boldsymbol{H}_{[,i]} \cdot \boldsymbol{s} + \overline{\boldsymbol{H}}_{[,i]} \cdot \overline{\boldsymbol{s}} = \sum_{\ell=1}^{n-k} (h_{\ell,i} s_\ell + (1 - h_{\ell,i})(t - s_\ell))) \tag{6}$$

With this definition at hand we can outline a procedure that outputs the column permutation corresponding to sorting the columns with respect to the score function.

Algorithm 5 Sort function

```
function SORT(H, s, t)
    for i ← 1, n do
        compute ψ_i(s) with Equation (6)
    Π ← sort ψ_i(s) in descending order
    return Π
```

Now, the question here is whether this procedure manages to discriminate between column belonging to the support of $\boldsymbol{e}$, where the t errors lie, and the rest of the columns of $\boldsymbol{H}$. The following result proves that there is a score difference between column indices in $\mathsf{Supp}(\boldsymbol{e})$ and column indices outside of $\mathsf{Supp}(\boldsymbol{e})$. This is a result taken from [FL20] and reformulated with respect to our formalism.

Lemma 3. *Let $\boldsymbol{H} \in \{0,1\}^{(n-k)\times n}$ be a matrix, with distribution given by $h_{j,i} \sim \mathcal{B}er(\frac{1}{2})$ and $\boldsymbol{s} \in \mathbb{N}^{n-k}$ such that $\exists\, \boldsymbol{e} \in \{0,1\}^n$ with $HW(\boldsymbol{e}) = t$ satisfying $\boldsymbol{H}\boldsymbol{e} = \boldsymbol{s}$. Then $\psi_i(\boldsymbol{s})$ is a random variable that follows the distribution*

$$\psi_i(\boldsymbol{s}) \sim \begin{cases} \mathcal{B}((n-k)t, \frac{1}{2}) & , i \notin \mathsf{Supp}(\boldsymbol{e}) \\ n-k+\mathcal{B}((n-k)(t-1), \frac{1}{2}) & , i \in \mathsf{Supp}(\boldsymbol{e}) \end{cases} \tag{7}$$

Notice that, for $i \notin \mathsf{Supp}(\boldsymbol{e})$ the most expected value of $\psi_i(\boldsymbol{s})$ would be around $(n-k)t/2$, as for $i \in \mathsf{Supp}(\boldsymbol{e})$ it is more likely to have $\psi_i(\boldsymbol{s})$ concentrated around $(n-k)/2 + (n-k)t/2$. Typically there is a gap between the average values equal to $(n-k)/2$, which enables us to discriminates between positions $i \in \mathsf{Supp}(\boldsymbol{e})$ and $i \notin \mathsf{Supp}(\boldsymbol{e})$. Notice that the gap between these two expected values is increasing in $n-k$. This fact suggests that the probability of discriminating indices in the support of the error vector increases when $n-k$ increases.

The syndrome in $\mathbb{N}$, which might be partially incorrect as detailed in Section 3.4 is only used up to this point, to identify a good permutation.

4.1.2 Score function in the presence of noise

Let us see how the score function behaves in the presence of noise. We model the noise on the syndrome vector as a symmetric random variable and give some particular cases. Let us denote the faulty syndrome by $\widetilde{s} = s + \epsilon$, where ϵ denotes the random variable describing the faults.

Proposition 1. *For any $j \in \mathbb{N}^*_{n-k}$ let ϵ_j be a discrete random variable defined over the set $\mathbb{Z}_{-d,d}$ satisfying:*

- *ϵ_j are independent and identically distributed*
- $\mathsf{Prob}(\epsilon_j = \alpha) = \mathsf{Prob}(\epsilon_j = -\alpha)$ *for all $\alpha \in \mathbb{Z}_{-d,d}$ (symmetry).*
- $\mathsf{Prob}\left((\epsilon_j = \alpha) \cap (h_{j,i} = \beta)\right) = \mathsf{Prob}(\epsilon_j = \alpha)\mathsf{Prob}(h_{j,i} = \beta)$ *for any $\alpha \in \mathbb{Z}_{-d,d}, \beta \in \{0,1\}$ (independence with $h_{j,i}$).*

Then for all $i \in \mathbb{N}^*_n$ we have $\mathsf{Prob}\left(\psi_i(\widetilde{\boldsymbol{s}}) - \psi_i(\boldsymbol{s}) = \alpha\right) = \mathsf{Prob}\left(\sum_{j=1}^{n-k} \epsilon_j = \alpha\right)$.

One can deduce from Proposition 1 that the value of the score function on the faulty syndrome deviates from the correct value identically for positions $i \in \mathsf{Supp}(\boldsymbol{e})$ and $i \notin \mathsf{Supp}(\boldsymbol{e})$. Moreover, if the sum of ϵ_j is centred around zero and has a relatively small standard deviation compared to that of $\mathcal{B}((n-k)t, 1/2)$ then the faulty syndrome has little effect on the sorting method shown in Algorithm 5. In other words, ψ can tolerate such errors, and can discriminate positions in the support of $\boldsymbol{e}$ from positions outside of it.

We will now consider the exact value of the syndrome $\boldsymbol{s}^* = \boldsymbol{s} \mod 2 \in \mathbb{F}_2^{n-k}$ which is the *correct* ciphertext. Moving forward, there are two direct algorithms for retrieving the error positions using the permutation $\boldsymbol{\Pi}$ returned by $\textsc{sort}(\boldsymbol{H}, \boldsymbol{s}, t)$.

4.2 ISD-based algorithms that exploit the good permutation

4.2.1 t-Threshold Score Decoder

Let us suppose that the permutation $\boldsymbol{\Pi}$ arranges the bit positions of $\boldsymbol{e}$ in such a manner that it can be split into two sub-vectors, the left part is the all-ones vector, and the right part is the all-zeroes vector. If this is the case, an algorithm only checks if the sum modulo 2 of the first t columns of $\boldsymbol{H\Pi}$ equals $\boldsymbol{s}^*$. If the condition is satisfied it returns a solution $\boldsymbol{e} = \boldsymbol{\Pi}(\mathbf{1}_t \parallel \mathbf{0}_{n-t})^t$. This means that the identified permutation is "perfect", and is capable of bringing t ones in the first t positions of the error vector.

Remark 1. Under the assumption that computing the product of two integers runs in $\mathcal{O}(1)$, the worst case time complexity of Algorithm 5 is $\mathcal{O}((n-k)n) = \mathcal{O}(n^2)$. Indeed, notice that computing the list of values $\psi_i(\boldsymbol{s})$ is equivalent to:

$$[\psi_i(\boldsymbol{s})]_{i \in \mathbb{N}^*_n} = \boldsymbol{s}^t\boldsymbol{H} + \overline{\boldsymbol{s}}^t\overline{\boldsymbol{H}}. \tag{8}$$

Hence, the overall time complexity of Algorithm 5 reduces to three matrix-vector multiplications, i.e., $\boldsymbol{s}^t\boldsymbol{H}$, $\overline{\boldsymbol{s}}^t\overline{\boldsymbol{H}}$ and $\boldsymbol{H\Pi}(\mathbf{1}_t \parallel \mathbf{0}_{n-t})^t$, and sorting a list of n integers.

Proposition 2. *`t-Threshold Score Decoder` outputs a valid solution as long as* $\min\{\psi_i(\boldsymbol{s}), i \in \mathsf{Supp}(\boldsymbol{e})\} > \max\{\psi_i(\boldsymbol{s}), i \in \mathbb{N}^*_n \setminus \mathsf{Supp}(\boldsymbol{e})\}$.

Proof. If $\min\{\psi_i(\boldsymbol{s}), i \in \mathsf{Supp}(\boldsymbol{e})\} > \max\{\psi_i(\boldsymbol{s}), i \in \mathbb{N}^*_n \setminus \mathsf{Supp}(\boldsymbol{e})\}$ holds this implies that $\boldsymbol{e}^t\boldsymbol{\Pi} = (\mathbf{1}_t \parallel \mathbf{0}_{n-t})$. Hence, we have $\boldsymbol{s}^* = \boldsymbol{He} = \boldsymbol{H\Pi\Pi}^{-1}\boldsymbol{e} = \boldsymbol{H\Pi}(\boldsymbol{e}^t\boldsymbol{\Pi})^t = \boldsymbol{H\Pi}(\mathbf{1}_t \parallel \mathbf{0}_{n-t})^t = (\boldsymbol{H\Pi})\boldsymbol{e}'$, which ends the proof. □

Our simulations have shown that, unfortunately, there are many cases where the positions in $\mathsf{Supp}(\boldsymbol{e})$ are not mapped on the first t positions by the permutation $\boldsymbol{\Pi}$, but rather on the first r positions with $r > t$. To deal with this situation we propose the following more powerful procedure.

4.2.2 Rank-Threshold Score Decoder

Let us suppose that the permutation $\boldsymbol{\Pi}$ arranges the bit positions of $\boldsymbol{e}$ in such a way that, on the last $n-r$ positions of the permuted vector, all the values are equal to zero. In addition, if $\boldsymbol{H\Pi}$ has column rank r on the first r columns, then one can retrieve a solution by computing a partial Gaussian elimination, i.e., using the reduced row-echelon form of $\boldsymbol{H\Pi}$ as follows.

First, use the $\text{SORT}(\boldsymbol{H}, \boldsymbol{s}, t)$ function to obtain a permutation $\boldsymbol{\Pi}$ for $\boldsymbol{H}$. Then, use the syndrome $\boldsymbol{s}^* = \boldsymbol{s} \mod 2 \in \mathbb{F}_2^{n-k}$, $\boldsymbol{\Pi}$ and $\boldsymbol{H}$ to retrieve $\boldsymbol{e}$ by means of linear algebra. We start by computing the reduced row-echelon form of $\boldsymbol{A}^*, \boldsymbol{H}^* \leftarrow \texttt{rref}(\boldsymbol{H\Pi})$. If $\text{HW}(\boldsymbol{A}^*\boldsymbol{s}^*)$ is equal to t then we get $\boldsymbol{e} = \boldsymbol{\Pi}(\boldsymbol{A}^*\boldsymbol{s}^* \parallel \boldsymbol{0}_k)^t$.

Remark 2. The procedure $\texttt{rref}(\boldsymbol{H\Pi})$ is equivalent to performing a partial Gaussian elimination over $\mathbb{F}_2$. Indeed, there is an $(n-k) \times (n-k)$ non-singular matrix $\boldsymbol{A}^*$ such that, $\boldsymbol{A}^*\boldsymbol{H\Pi} = \begin{bmatrix} \boldsymbol{I_r} \\ \boldsymbol{0_{n-k-r,r}} \end{bmatrix} \parallel \boldsymbol{B}^*$ where $\boldsymbol{H\Pi} = [\boldsymbol{A} \parallel \boldsymbol{B}]$ with $\boldsymbol{A}$ a $(n-k) \times r$ matrix satisfying $\boldsymbol{A}^*\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I_r} \\ \boldsymbol{0_{n-k-r,r}} \end{bmatrix}$, and $\boldsymbol{B}^* = \boldsymbol{A}^*\boldsymbol{B}$.

From the description of the algorithm above, the following result can be deduced.

Proposition 3. *`Rank-Threshold Score Decoder` outputs a valid solution as long as there exists at least one set* $L \subset \mathbb{N}_n^* \setminus \mathsf{Supp}(\boldsymbol{e})$ *with* $\#L \geq n-r$ *such that* $\min\{\psi_i(\boldsymbol{s}), i \in \mathsf{Supp}(\boldsymbol{e})\} > \max\{\psi_i(\boldsymbol{s}), i \in L\}$.

The overall time complexity of the `Rank-Threshold Score Decoder` is $\mathcal{O}((n-k)^3)$, since it is dominated by the partial Gaussian elimination, i.e., the computation of $\boldsymbol{A}^*$.

Notice that a valid permutation for the `Rank-Threshold Score Decoder` is also a valid permutation for the `t-Threshold Score Decoder`.

`Rank-Threshold Score Decoder` uses linear algebra in the same way as the Prange decoder does for syndrome decoding [Pra62]. Hence, it is natural to further explore the improvements of the Prange decoder to improve the success of the decoder. Hence, we propose to extend `Rank-Threshold Score Decoder` so that it covers error vectors with a more general pattern. More precisely, instead of having permuted error vectors with all-zero vector on the last $n-r$ positions, we allow for δ non-zero positions. This is the principle of the Lee-Brickell decoder, shown in Algorithm 6.

Algorithm 6 Lee-Brickell Score Decoder

function LEE-BRICKELL SCORE DECODER($\boldsymbol{H}, \boldsymbol{s}^*, t$)
  Compute $\boldsymbol{\Pi} \leftarrow \text{SORT}(\boldsymbol{H}, \boldsymbol{s}, t)$
  Set $\boldsymbol{H\Pi} = [\boldsymbol{A} \parallel \boldsymbol{B}]$
  Compute $\boldsymbol{A}^*, \boldsymbol{H}^* \leftarrow \texttt{rref}(\boldsymbol{H\Pi})$ and $\boldsymbol{B}^* = \boldsymbol{A}^*\boldsymbol{B}$
  Compute $\boldsymbol{s}' = \boldsymbol{A}^*\boldsymbol{s}^*$
  if $\text{HW}(\boldsymbol{s}') == t$ **then**
    return $\boldsymbol{e} = \boldsymbol{\Pi}(\boldsymbol{s}' \parallel \boldsymbol{0}_k)^t$
  else
    for $i \leftarrow 1, \delta$ **do**
      S = `Gener-Subsets`($\{1, \ldots, n-r\}$,i)
      for E in S **do**
        $\boldsymbol{e}'' \leftarrow \texttt{Vector}(\{0,1\}, n-r, E)$
        $\boldsymbol{e}' \leftarrow \boldsymbol{s}' - \boldsymbol{B}^*\boldsymbol{e}''$
        if $\text{HW}(\boldsymbol{e}') == t - i$ **then**
          return $\left(\boldsymbol{\Pi}(\boldsymbol{e}' \parallel \boldsymbol{e}'')^t, \boldsymbol{\Pi}\right)$

Remark 3. The working factor of Algorithm 6 is given by

$$O((n-k)^3) + O\left(\binom{n-r}{\delta}(n-r)^2\right), \tag{9}$$

where the first term corresponds to the work factor of the $\texttt{rref}(\boldsymbol{H}\boldsymbol{\Pi})$, and the second term to the work factor of the computation of $\boldsymbol{e}'$ multiplied by the number of times $\boldsymbol{e}'$ is computed. For fixed parameters (n, k, t), one can estimate the value of δ that leads to a success probability close to 1. As we shall see in the experimental section, for all the cryptographic parameters in the *Classic McEliece* cryptosystem, $\delta = 2$ suffices to have at least 99.5 % of success. Thus, for these particular parameters, i.e., $\delta = 2, k = n/c$ where c is a constant, we have a worst case complexity of $O(n^4)$.

5 Experimental results

5.1 Side-channel measurements

All the side-channel measurements are power consumption measurements. They were performed using the ChipWhisperer platform [OC14]. The targets microcontroller that computes the syndrome is based on an ARM® Cortex®-M4 core, which is the hardware platform recommended by the NIST, and already used in related work such as the PQM4 libray [KRSS] to benchmark and test implementations of post-quantum cryptography algorithms. Figure 3 shows a side-channel trace measured during the syndrome computation.

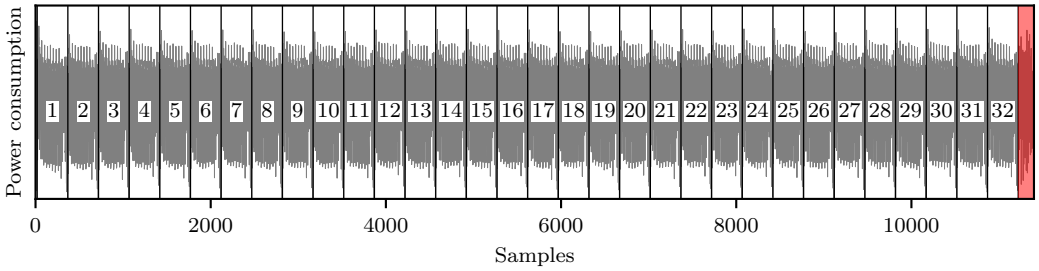


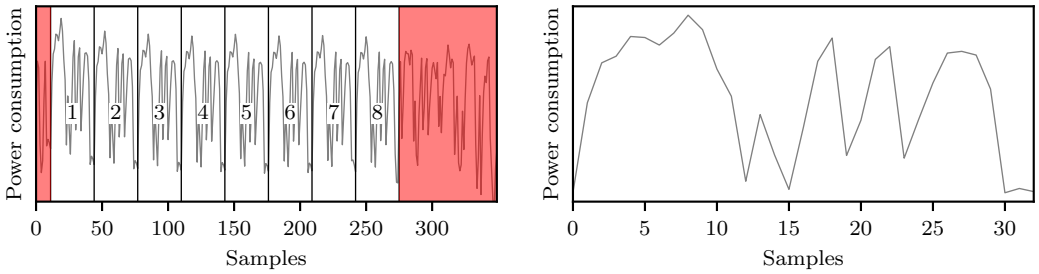
Figure 3: Side-channel trace of the syndrome computation ($n = 64$ for readability reasons)

For readability reasons, a very small value of $n = 64$ ($k = 32$, $t = 8$) is used for the plots in Figures 3 and 4, instead of the *Classic McEliece* parameters. A pattern is clearly visible and repeated $n - k = 32$ times. Samples under the red area are not useful for the analysis and are discarded later in the preprocessing step described in Section 5.2.

For the rest of the experiments, the values of n, k and t we considered are given in Table 1. It is worth noting that, for those parameters, side-channel measurements cannot be performed on the ChipWhisperer platform, for two reasons. First, the $\boldsymbol{H}$ matrix does not fit in the available memory of the micro-controller we used, which has 256 kB of Flash memory and 40 kB of RAM. Second, the number of samples which can be recorded, less than 100 000, is greatly exceeded by the duration of the full matrix-vector multiplication. For these two reasons, we perform the matrix-vector multiplication one row after the other, and then concatenate the recorded traces to obtain the side-channel trace for the full matrix-vector multiplication. This does not alter the attack scenario, since the attack works with the Hamming weight over rows independently to determine the value of the syndrome entry in $\mathbb{N}$. This could be fixed by using a more advanced experimental setup.

5.2 Side-channel traces preprocessing

The raw trace from Figure 3 is reshaped one time to obtain the trace shown in Figure 4a, which shows one of the 32 patterns of Figure 3. Each pattern corresponds to the multiplication of one row of the matrix $\boldsymbol{H}$ with the error vector $\boldsymbol{e}$. This reshaping is easily done by visually observing the width of a pattern.



(a) Trace after the first reshaping step

(b) Trace after the second reshaping step

Figure 4: Side-channel traces after each of the two reshaping steps.

Then, the trace from Figure 4a is reshaped a second time to obtain the trace shown in Figure 4b, which shows one of the eight patterns highlighted in Figure 4a. Each pattern corresponds to the multiplication of one byte of the matrix $\boldsymbol{H}$ with one byte of the error vector $\boldsymbol{e}$. Samples that come after the eighth pattern in Figure 4a, under the red area, are discarded since they are not useful in the derivation of the syndrome in $\mathbb{N}$. They correspond to the last instructions in the outer `for` loop of Algorithm 4, namely the exclusive-OR folding, the LSB extraction and the bit packing operations.

As claimed above in Section 3, the Hamming weight leakage model is much stronger than the Hamming distance leakage model in the recorded side-channel traces. This is shown in Figure 5, where the signal-to-noise ratio is plotted for both leakage models and superimposed on the average trace found in the preprocessed traces. The signal-to-noise ratio is computed by an F-test, that is, the ratio between the inter-class variability and the intra-class variability. As depicted, the maximum value for the F statistic is 54 times larger for the Hamming weight than for the Hamming distance leakage model, indicating that the leakage for the former is much stronger than for the latter.

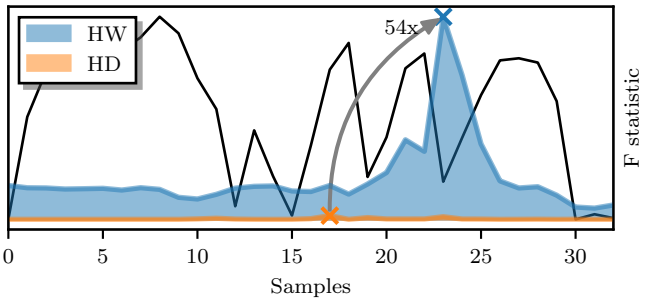


Figure 5: Signal-to-noise ratio for Hamming weight (HW) and Hamming distance (HD) leakages, superimposed with the power consumption data from Figure 4b.

The last preprocessing step consists in reducing the dimension of the data so that it is easier to handle by the classifier. We used `discriminant_analysis.LinearDiscriminantAnalysis`[2] from the `sklearn` Python package to map the preprocessed traces, possibly extended by considering adjacent rows with the method detailed in Section 3.3, to an 8-dimensional space by Linear Discriminant Analysis. The output space has eight dimensions since there are nine possible values for the Hamming weight of a byte. After this last preprocessing step, the traces are fed to the classifier which will be responsible for recovering the Hamming weight values.

[2] https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

5.3 Hamming weight recovery with a random forest

We used the implementation from `RandomForestClassifier`[3] of a random forest classifier from the `sklearn` Python package. We kept the parameters to their default values. In particular, there are one hundred of trees whose votes are combined by majority voting. We refer the reader to the documentation of the function for the other parameters.

For each (n, k, t) set of parameters, ten independent experiments were conducted. We restrict ourselves to ten experiments for each value of n because it takes a lot of time to perform the side-channel measurements for all the matrix rows. Ten classifiers were then trained and their accuracy computed from the test set. The accuracy we obtained was always very high, ranging from 99.17 % to 99.91 %, with $\Delta = 3$. With $\Delta < 3$, the accuracy is lower. With $\Delta > 3$, the dimensionality-reduction step exceeds the memory capacity of the computer we use (32 GB).

It is worth noting that, since we use only one trace for the training set, the number of training samples is different for each set of parameters. The classifier trained for $n = 3488$ sees a bit more than one million samples, while for $n = 8192$, almost seven million samples are available. This might explain why the classifier accuracy is higher and more consistent for the largest values of n. The detailed accuracy and number of training samples for each set of parameters are given in Table 3.

Table 3: Number of training samples and classifier accuracy for each *Classic McEliece* parameters set

Parameters set	`348864`	`460896`	`6688128`	`8192128`
n	3488	4608	6688	8192
k	2720	3360	5024	6528
t	64	96	128	128
# training samples	1 185 920	1 935 360	4 200 064	6 684 672
Classifier accuracy	0.9964 ($\sigma = 0.0021$)	0.9983 ($\sigma = 0.00092$)	0.9963 ($\sigma = 0.00076$)	0.9983 ($\sigma = 0.00039$)

5.4 Derivation of the approximate syndrome in $\mathbb{N}$

After the Hamming weight information has been recovered by the classifier, the syndrome in $\mathbb{N}$ is derived thanks to Equation (3) and the parity-correction step. For each parameters set, we compute the distribution of the errors made when estimating the syndrome values in $\mathbb{N}$. These distributions are shown in Figure 6 and are obtained by comparing the $(n - k)$ recovered and correct syndrome entries in $\mathbb{N}$ for each of the ten independent experiments.

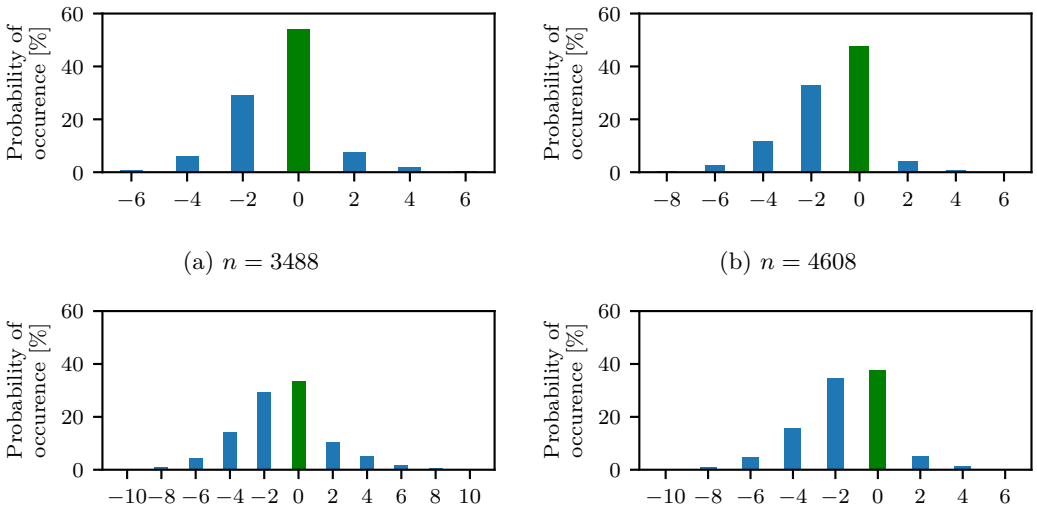
We can observe that, for all values of n, in most of the cases, the value of the syndrome entry in $\mathbb{N}$ is correctly recovered, as depicted by a green bar.

Another significant case is when the recovered value is equal to the real one minus two. This occurs when the Hamming weight of the intermediate value remains the same, but two bits inside actually flipped in opposite directions. For instance, the intermediate value b can change from `0b00000001` to `0b00000010`. The Hamming distance is two in this case, but is considered to be zero when looking at the Hamming weights only. This has been shown before in Figure 2b. Even though this happens very rarely, this is still visible as a result of the summation over large n values.

5.5 Score function evaluation on perfect and noisy syndromes in $\mathbb{N}$

After obtaining the syndrome in $\mathbb{N}$, we now assess the efficiency of the score function, that is, its ability to identify the error positions in the error vector.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html



(a) $n = 3488$ (b) $n = 4608$

(c) $n = 6688$ (d) $n = 8192$

Figure 6: Difference between the real and recovered syndrome entries in $\mathbb{N}$

The first type of simulations are done using *perfect* syndromes in $\mathbb{N}$. More exactly, given a public parity-check matrix $\boldsymbol{H}$ we randomly generated a set of 10 000 error vectors and computed their corresponding syndromes in $\mathbb{N}$. For each syndrome we we computed the success probability of the Lee-Brickell Score Decoder for all the *Classic McEliece* parameters with respect to the value of δ.

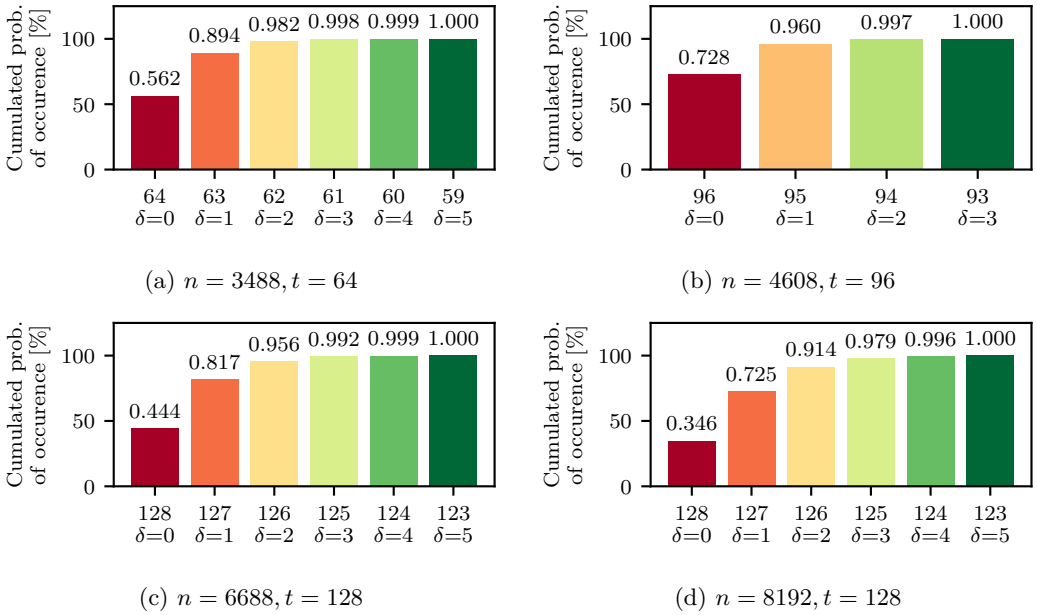
Table 4: Success probability of the Lee-Brickell Score Decoder for the *Classic McEliece* with *perfect* syndromes in $\mathbb{N}$.

Parameters set	n	k	t	$\mathbb{N}-$SDP decoder				
				$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$
348864	3488	2720	64	0.773	0.977	0.999	1.000	1.000
460896	4608	3360	96	0.860	0.991	0.999	1.000	1.000
6688128	6688	5024	128	0.780	0.980	1.000	1.000	1.000
8192128	8192	6528	128	0.661	0.943	0.995	0.999	1.000

Notice that, in the case of perfect syndromes in $\mathbb{N}$, $\delta = 2$ gives a probability of success greater than or equal to 0.995 for all the parameters. In order to increase it up to 0.999 taking $\delta = 3$ is sufficient.

For the second type of simulations, we generated noisy syndromes in $\mathbb{N}$ by exploiting the empirical error distributions shown in Figure 6. This allows us to estimate the efficiency of the score function more accurately. Here, we perform again 10 000 simulations for each parameters set. Experimental results are shown in Figure 7. We observe that a larger δ value is necessary to reach the same success probability, when compared to the values given in Table 4. This makes sense since, when dealing with experimental data, the recovered syndrome in $\mathbb{N}$ might not be perfectly recovered.

Overall, however, we can observe that a very high success probability is achieved for small values of δ.



(a) $n = 3488, t = 64$ (b) $n = 4608, t = 96$

(c) $n = 6688, t = 128$ (d) $n = 8192, t = 128$

Figure 7: Probability of finding at least x ones in the first $(n - k)$ positions, obtained from *noisy* syndromes in $\mathbb{N}$, derived from the distributions in Figure 6.

6 Countermeasures

The main weakness of the *Classic McEliece* cryptosystem, that makes it vulnerable to the proposed attack, is that the error weight t is small. Masking can be used to instead have an error vector with a more general error pattern.

6.1 Classical masking

The leakage used to obtain the values of the syndrome in $\mathbb{N}$ instead of $\mathbb{F}_2$ can be hidden with a mask. Indeed, the attack succeeds because the message has a low weight $t \approx \sqrt{n}$.

Using vectors of random Hamming weight $\frac{n}{2}$ limits the accuracy of the random forest method to recover the Hamming weights. As a consequence, the derived syndrome in $\mathbb{N}$ is "more incorrect". Therefore, the permutation $\boldsymbol{\Pi} \leftarrow \textsc{sort}(\boldsymbol{H}, \boldsymbol{s}, t)$ is not suitable and we must resort to exhaustive enumeration which is computationally expensive.

The proposed masking scheme is as follows. We first draw a random value $x \in \mathbb{F}_2^{n-k}$ and add it to $\boldsymbol{e}$. Then, two matrix-vector products are performed: $\boldsymbol{H}_{\text{pub}}(\boldsymbol{e} \oplus x)$ and $\boldsymbol{H}_{\text{pub}}x$. Since the operations are linear, we have $\boldsymbol{H}_{\text{pub}}\boldsymbol{e} = \boldsymbol{H}_{\text{pub}}(\boldsymbol{e} \oplus x) \oplus (\boldsymbol{H}_{\text{pub}}x)$. This is shown in Algorithm 7.

Algorithm 7 Encryption using classical masking

1: **function** ENCRYPT_MASKED($\boldsymbol{m}$, pk)
2: Encode $\boldsymbol{m} \to \boldsymbol{e}$ with $\text{HW}(\boldsymbol{e}) = t$
3: Randomly pick an element $\boldsymbol{x} \in \mathbb{F}_2^n$ ▷ $\text{HW}(\boldsymbol{x}) \approx \frac{n}{2}$
4: Compute $\boldsymbol{s} = \boldsymbol{H}_{\text{pub}}(\boldsymbol{e} \oplus x) \oplus \boldsymbol{H}_{\text{pub}}x = \boldsymbol{H}_{\text{pub}}\boldsymbol{e}$
5: **return** $\boldsymbol{s}$

The overhead of such a masked implementation is linear in time and randomness. Both matrix-vector products are performed with vectors of average Hamming weight $\frac{n}{2}$.

Performing the attack twice, one on $\boldsymbol{H}_{\text{pub}}(\boldsymbol{e} \oplus x)$ and one on $\boldsymbol{H}_{\text{pub}}x$, and combining the results is unlikely to work since the average Hamming weight is now $\frac{n}{2}$ instead of $t \approx \sqrt{n} << \frac{n}{2}$.

Another way to attack is to directly perform a second-order attack. In that case, we know that masking theoretically increases the difficulty of an attack exponentially in the number of shares [PR13].

6.2 Masking with a codeword

When applying masking during the encryption, we can note that, if the mask belongs to the dual code of the public matrix, the mask will be automatically removed during the encryption process. It is what we propose here in Algorithm 8.

Algorithm 8 Encryption using a random codeword

1: **function** ENCRYPT_MASKED_WITH_CODEWORD($\boldsymbol{m}$, pk)
2: Encode $\boldsymbol{m} \to \boldsymbol{e}$ with $\text{HW}(\boldsymbol{e}) = t$
3: Compute the dual matrix $\boldsymbol{G}_{\text{pub}}$ of the public matrix $\boldsymbol{H}_{\text{pub}}$
4: Randomly pick an element $\boldsymbol{x} \in \mathbb{F}_2^k$
5: Compute $\boldsymbol{e}' = \boldsymbol{x}\,\boldsymbol{G}_{\text{pub}} \oplus \boldsymbol{e}$ ▷ $\text{HW}(\boldsymbol{e}') \approx \frac{n}{2}$
6: Compute $\boldsymbol{s} = \boldsymbol{H}_{\text{pub}}\boldsymbol{e}' = \boldsymbol{H}_{\text{pub}}(\boldsymbol{x}\,\boldsymbol{G}_{\text{pub}} \oplus e) = \boldsymbol{H}_{\text{pub}}\boldsymbol{e}$
7: **return** $\boldsymbol{s}$

The computation of the dual matrix $\boldsymbol{G}_{\text{pub}}$ of the public matrix $\boldsymbol{H}_{\text{pub}}$ is straightforward if $\boldsymbol{H}_{\text{pub}}$ is in systematic form. The vector $\boldsymbol{e}' \in \mathbb{F}_2^{n-k}$ has an average weight of $\frac{n}{2}$. Since $\boldsymbol{G}_{\text{pub}}$ is indistinguishable from a random binary matrix, each bit of $x\boldsymbol{G}_{\text{pub}}$ can be seen as a random bit. There exists 2^k different masks $x\boldsymbol{G}_{\text{pub}}$. The large number of mask candidates prevents from exhaustive search over all the possible mask values.

It is worth noting that the two masking techniques presented here do not need any unmasking during the decryption process. Therefore, they incur an overhead only during the encryption process.

7 Conclusion

This article presents a message recovery attack against the *Classic McEliece* cryptosystem using side-channel analysis on the power consumption and machine learning techniques. The attack works in two steps. First, we retrieve the value of the syndrome in $\mathbb{N}$, during the encryption process, using only *one* trace. Second, we recover the secret message $\boldsymbol{m}$ using a new algorithm based on a computationally-efficient score function and known information-set decoding methods. We showed that the solver we designed is tolerant to errors in the syndrome recovery process which might stem from side-channel analysis inaccuracy. This attack, although is follows the same attack path as a previously proposed message-recovery attack based on laser fault injection, improves it in many ways and makes it much more practical and efficient overall.

We can identify several research perspectives to continue this work. First, an interesting aspect to consider would be the influence of the classification accuracy on the overall attack success. Indeed here, the classifier, although quite simple, is very accurate, with an accuracy of more than 99.5 %. Evaluating how a less precise classifier would affect the attack success would be relevant.

Another research direction could be to further improve the derivation of the syndrome in $\mathbb{N}$. Indeed, while we considered only the Hamming weight information in this work, the Hamming distance could be exploited as well. Further studies could focus on combining those two leakages in an efficient manner.

Finally, although we proposed two countermeasures that are applicable to the syndrome computation, a thorough evaluation of their cost and efficiency remains to be done.

Acknowledgements

V-F. Drăgoi was supported by a grant of the Ministry of Research, Innovation and Digitization, CNCS/CCCDI – UEFISCDI, project number PN-III-P1-1.1-PD-2019-0285, within PNCDI III.

This work is supported by the French National Research Agency in the framework of the "Investissements d'avenir" program "ANR-15-IDEX-02" and the LabEx PERSYVAL "ANR-11-LABX-0025-01".

References

[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.

[BCS13] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 250–272. Springer, Heidelberg, August 2013.

[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, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, Heidelberg, April 2012.

[BM18] Leif Both and Alexander May. Decoding linear codes with high error rate and its impact for LPN security. In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, pages 25–46. Springer, Heidelberg, 2018.

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

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[BSNK19] Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri. NIST post-quantum cryptography- A hardware evaluation study. Cryptology ePrint Archive, Report 2019/047, 2019. https://eprint.iacr.org/2019/047.

[CC98] Anne Canteaut and Florent Chabaud. A new algorithm for finding minimum-weight words in a linear code: Application to mceliece's cryptosystem and to narrow-sense BCH codes of length 511. *IEEE Trans. Inf. Theory*, 44(1):367–378, 1998.

[CC21] Ming-Shing Chen and Tung Chou. Classic mceliece on the ARM cortex-M4. *IACR TCHES*, 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, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 438–467. Springer, Heidelberg, October 2021.

[Cho18] Tung Chou. McBits revisited: toward a fast constant-time code-based KEM. *Journal of Cryptographic Engineering*, 8(2):95–107, June 2018.

[DCC+20] Vlad-Florin Dragoi, Pierre-Louis Cayrel, Brice Colombier, Dominic Bucerzan, and Sorin Hoara. Solving a modified syndrome decoding problem using integer programming. *International Journal of Computers Communications & Control*, 15(5), 2020.

[DFA+20] Viet Ba Dang, Farnoud Farahmand, Michal Andrzejczak, Kamyar Mohajerani, Duc Tri Nguyen, and Kris Gaj. Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches. Cryptology ePrint Archive, Report 2020/795, 2020. https://eprint.iacr.org/2020/795.

[DK20] Julian Danner and Martin Kreuzer. A fault attack on the niederreiter cryptosystem using binary irreducible goppa codes. *Journal of Groups, Complexity, Cryptology*, 12(1):1–20, 2020.

[Dor43] Robert Dorfman. The detection of defective members of large populations. *Annals of Mathematical Statistics*, 14:436–440, 1943.

[Dum89] Ilya Dumer. Two decoding algorithms for linear codes. *Probl. Inform. Transm.*, 25(1):17–23, 1989.

[EGHP09] Thomas Eisenbarth, Tim Güneysu, Stefan Heyse, and Christof Paar. MicroEliece: McEliece for embedded devices. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 49–64. Springer, Heidelberg, September 2009.

[EMZ21] Andre Esser, Alexander May, and Floyd Zweydinger. Mceliece needs a break – solving mceliece-1284 and quasi-cyclic-2918 with modern isd. Cryptology ePrint Archive, Report 2021/1634, 2021. https://eprint.iacr.org/2021/1634.pdf.

[FL20] Uriel Feige and Amir Lellouche. Quantitative group testing and the rank of random matrices. *CoRR*, abs/2006.09074, 2020.

[FS09] Matthieu Finiasz and Nicolas Sendrier. Security bounds for the design of code-based cryptosystems. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 88–105. Springer, Heidelberg, December 2009.

[GHKL19] Oliver Gebhard, Max Hahn-Klimroth, Dominik Kaaser, and Philipp Loick. Quantitative group testing in the sublinear regime. *CoRR*, abs/1905.01458, 2019.

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

[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.

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

[HPR+21] Anna-Lena Horlemann-Trautmann, Sven Puchinger, Julian Renner, Thomas Schamberger, and Antonia Wachter-Zeh. Information-set decoding with hints. Cryptology ePrint Archive, Report 2021/279, 2021. https://eprint.iacr.org/2021/279.

[KRF+21] Vatistas Kostalabros, Jordi Ribes-González, Oriol Farràs, Miquel Moretó, and Carles Hernández. Hls-based HW/SW co-design of the post-quantum classic mceliece cryptosystem. In *International Conference on Field-Programmable Logic and Applications1*, pages 52–59, Dresden, Germany, August 2021. IEEE.

[KRSS] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[KRSS19] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. pqm4: Testing and benchmarking NIST PQC on ARM cortex-M4. Cryptology ePrint Archive, Report 2019/844, 2019. https://eprint.iacr.org/2019/844.

[LB88] Pil Joong Lee and Ernest F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 275–280. Springer, Heidelberg, May 1988.

[LCPR19] Kangwook Lee, Kabir Chandrasekher, Ramtin Pedarsani, and Kannan Ramchandran. SAFFRON: A fast, efficient, and robust framework for group testing based on sparse-graph codes. *IEEE Transactions on Signal Processing*, 67(17):4649–4664, 2019.

[Leo88] Jeffrey S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Trans. Inf. Theory*, 34(5):1354–1359, 1988.

[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, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 881–910. Springer, Heidelberg, December 2020.

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

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

[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, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 203–228. Springer, Heidelberg, April 2015.

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

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

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Heidelberg, May 2013.

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

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

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

[Ste88] Jacques Stern. A method for finding codewords of small weight. In *Proceedings of the 3rd International Colloquium on Coding Theory and Applications*, page 106–113, Berlin, Heidelberg, Nov 1988. Springer-Verlag.

[TS16] Rodolfo Canto Torres and Nicolas Sendrier. Analysis of information set decoding for a sub-linear error weight. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings*, volume 9606 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2016.

[vMHG16] Ingo von Maurich, Lukas Heberle, and Tim Güneysu. IND-CCA secure hybrid encryption from QC-MDPC niederreiter. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography - 7th International Workshop, PQCrypto 2016*, pages 1–17. Springer, Heidelberg, 2016.

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

[XIU+21] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. Fault-injection attacks against nist's post-quantum cryptography round 3 KEM candidates. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 33–61. Springer, 2021.