Profiled Side-channel Attack on Cryptosystems based on the Binary Syndrome Decoding Problem

Brice Colombier[*], Vlad-Florin Drăgoi[†‡], Pierre-Louis Cayrel[§], Vincent Grosso[§]
[*]Univ Grenoble Alpes, CNRS, Grenoble INP, TIMA, 38000 Grenoble, France
brice.colombier@grenoble-inp.fr
[†]Faculty of Exact Sciences, Aurel Vlaicu University, Arad, Romania
vlad.dragoi@uav.ro
[‡]LITIS, University of Rouen Normandie, Saint-Etienne du Rouvray, France
[§]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, builds its security on the hardness of the syndrome decoding problem, which is a classic hard problem in code-based cryptography. This cryptosystem was recently targeted by a laser fault injection attack leading to message recovery. Regrettably, the attack setting is very restrictive and it does not tolerate any error in the faulty syndrome. Moreover, it depends on the very strong attacker model of laser fault injection, and does not apply to optimised implementations of the algorithm that make optimal usage of the machine words capacity. In this article, we propose a to change the 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, obtained with power consumption analysis, is sufficient to obtain an integer syndrome, as required by the attack framework. 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 and information-set decoding algorithms, to recover the message from the, possibly inaccurate, recovered integer syndrome. Finally, we present a masking countermeasure against the proposed attack.**

***Index Terms*—Post-quantum Cryptography, Syndrome Decoding Problem, Side-channel Attack**

I. Introduction

The binary syndrome decoding problem (SDP) is a foundational problem of code-based cryptosystems. Its NP-hardness was proven in 1978 [1] and is the basis of various cryptographic constructs like the Niederreiter public-key cryptosystem [2], the FSB hash function [3], the SYND stream cipher [4], the Stern identification scheme [5] or a pseudo-random number generator [6]. It is also the basis of several Key Encapsulation Mechanisms submitted to the NIST post-quantum cryptography standardisation process, like BIKE [7] or *Classic McEliece* [8]. The latter has been selected as a finalist for the third round, as announced on July 2020. It instantiates the Niederreiter cryptosystem [2] with binary Goppa codes, which is the dual of the McEliece cryptosystem [9]. The parameters of the cryptosystem are set with respect to the complexity of the best information-set decoding (ISD) attack strategy [10], [11], [12], [13], [14], 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 [15], [16], [17], [18], [19], [20], [21], [22], [23] as well as physical attacks, both side-channel [24] and fault injection attacks [25], [26], [27].

A message-recovery attack that uses side-channel leakage during the decryption process was proposed in [24]. The central idea of this article is to add random columns of the parity matrix to the syndrome. The modified syndrome is fed 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. ISD is leveraged to reduce the number of calls to the oracle.

In [27] the authors recover the message using laser-fault injection during the encryption process. By performing a bit-set on the instructions, the XOR instruction is turned into an ADD and the matrix-vector multiplication is performed over $\mathbb{N}$ instead of $\mathbb{F}_2$. A modified version of the syndrome decoding problem is derived from this and the message can then be recovered within minutes.

Some attacks aim at recovering the secret key instead. In [25], 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 [26] 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 [26]. However, it has not been applied to the *Classic McEliece* cryptosystem.

The idea of augmenting the syndrome decoding problem (SDP) with additional information obtained by physical attacks has been studied in [27], [28]. While in [28] the overall cost of the attack remains unfeasible practically, in [27], all the parameters set of *Classic McEliece* have been attacked. In the so-called "Integer Syndrome Decoding Problem" ($\mathbb{N}-\mathsf{SDP}$), introduced in [29], [27], the matrix-vector multiplication is performed over $\mathbb{N}$ instead of $\mathbb{F}_2$. Solutions to this problem can be found with Integer Linear Programming techniques such as the simplex [29] or interior-points methods [27]. The latter is very computationally efficient, and can solve the $\mathbb{N}-\mathsf{SDP}$ using only a small proportion of the parity-check rows. Authors show by simulations that the empirical complexity is $\mathcal{O}(n^2)$. However, this algorithm has several drawbacks: its exact complexity remains unknown, and more important, it does not tolerate any error in the integer syndrome. This is a major practical problem for physical attacks, where perfect repeatability during the fault injection process is rarely obtained.

It turns out that the $\mathbb{N}-\mathsf{SDP}$ had already been considered in the literature, first by Dorfman [30], and is known as the "group testing problem" or "quantitative group testing" [31], [32], [33]. Here, we make use of the algorithms from [33] and modify them to fit the context of side-channel analysis. However, further analysis is also needed to adapt these algorithms to the framework of ISD methods.

Contributions: In this article, a message-recovery side-channel attack against the *Classic McEliece* cryptosystem is performed. The side-channel attack is performed by power consumption analysis on the matrix-vector multiplication during the encryption process. By means of random forests, the value of the integer syndrome is recovered and is used later as input for the $\mathbb{N}-\mathsf{SDP}$. We then present an algorithm that can retrieve the initial message from the estimated integer syndrome in polynomial time in the code length.

First of all, we introduce a novel method to obtain the integer syndrome, as required by the attack framework presented in [27]. While the attack in [27] 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 integer syndrome from Hamming weight information obtained during the syndrome computation. Another significant difference, compared to laser fault injection attacks, is that no modification of the ciphertext is visible 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 a recovered integer syndrome, obtained by side-channel analysis.

Second, we develop a new way to recover the positions of the errors in the error vector (*i.e.*, the message) from the integer syndrome. The initial attack in [27] made use of integer linear programming solvers for this task. Although these solvers are quite efficient, they do not tolerate any error in the integer syndrome. 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 integer syndrome and the public key (*i.e.*, the 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. The permutation can 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 ISD algorithm. Hence, the algorithm we propose has two main steps. First, we use the recovered integer syndrome to find a "good" permutation. Second, we use it in addition to the binary syndrome (*i.e.*, the exact ciphertext) in an ISD variant to retrieve the initial message (*i.e.*, the binary error vector) solution to the syndrome decoding problem. We evaluate the resistance of this method to errors in the recovered integer syndrome 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 [27].

We have chosen to implement the attack against the *Classic McEliece* for several reasons. It is the single code-based candidate remaining in the final round of the NIST standardization process. It uses binary Goppa codes, which are up to now the most secure choice for public key encryption in terms of algebraic codes. Also, no efficient distinguisher for random binary Goppa codes exist, except for high rate code which are not considered in this context. Up to now, no efficient algorithm for retrieving the Goppa polynomial from the public data exist, fact that sustains the security of the scheme. However, we would like to point out that the proposed attack can be adapted to other Niederreiter-like schemes, like BIKE [7]. The main argument for this claim is that we are targeting the implementation of a matrix-vector multiplication, which is a building block for all code-based schemes based on the binary SDP. Also, we show that after performing the attack, the knowledge of the private message allows one to entirely dismantle the security of the key encapsulation mechanism (KEM).

Organisation: Section II introduces the syndrome decoding problem and the best known algorithms to solve it. We then consider the Niederreiter cryptosystem, which is a classic example of a cryptosystem built on the syndrome decoding problem, before moving on to implementations of code-based cryptosystems and existing physical attacks. Section III presents the proposed side-channel analysis method, based on machine learning, used to recover the integer syndrome. Section IV then introduces decoders that exploit the recovered integer syndrome to reveal the secret message. Experimental results are given in Section V while Section VI introduces a few ideas of countermeasures to thwart the proposed attack. Finally, we conclude in Section VII.

II. Code-based cryptosystems

A. Coding theory

a) 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 $\mathbb{N}_n^* = \{1,\ldots,n\}$ and $\mathbb{Z}_{-n,n} = \{-n,\ldots,0,\ldots,n\}$. Matrices and vectors are written in bold capital, respectively small letters, e.g., a vector of length n is $\boldsymbol{c} = (c_1,\ldots,c_n)$ and a $k \times n$ matrix is $\boldsymbol{H} = (h_{j,i})_{(j,i)\in\mathbb{N}_k^*\times\mathbb{N}_n^*}$. We 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 $\boldsymbol{H}$.

Algorithm 1 Niederreiter encryption

```
1: function ENCRYPT($\boldsymbol{m}$, $\boldsymbol{H}_{\text{pub}}$)
2:     Encode $\boldsymbol{m} \to \boldsymbol{e}$ with $\text{HW}(\boldsymbol{e}) = t$
3:     Compute $\boldsymbol{s}^* \leftarrow \boldsymbol{H}_{\text{pub}}\boldsymbol{e}$
4:     return $\boldsymbol{s}^*$
```

Binary vectors and matrices are packed into blocks of 8-bit sub-vectors and sub-matrices. For example, we write $\boldsymbol{c} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{n/8})$, where $\boldsymbol{c}_{i+1} = (c_{8i+1}, \ldots, c_{8i+8})$ for vectors, and $\boldsymbol{H}_{[i,j+1]}$ is an 8-bit vector defined as $\boldsymbol{H}_{[i,j+1]} = (h_{i,8j+1}, \ldots, h_{i,8j+8})$ for matrices. The set of all $k \times n$ matrices over $\mathbb{F}$ is $\mathbb{F}^{k\times n}$. The support of a vector 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 $\text{HW}(\boldsymbol{e})$ is the number of its non-zero coordinates. The Hamming distance between two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ is written as $\text{HD}(\boldsymbol{a}, \boldsymbol{b})$.

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 $\boldsymbol{G}\boldsymbol{H}^T = \boldsymbol{0}$. 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 (Binary syndrome decoding problem SDP)**.**
Inputs: $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$, $\boldsymbol{s}^* \in \mathbb{F}_2^{n-k}$, $t \in \mathbb{N}^*$
Output: $\boldsymbol{e} \in \mathbb{F}_2^n$ with $\text{HW}(\boldsymbol{e}) = t$, such that $\boldsymbol{H}\boldsymbol{e} = \boldsymbol{s}^*$.

The decisional version of the SDP, also known as the coset weight problem, belongs to the class of NP-complete problems [1]. Hence, all the existing algorithms for solving SDP are exponential in the code parameters [34], [35], [36], [37], [14]. Algorithms for solving the SDP include [38], [39], [40], [41], [42], [43], [44], [10], [11], [12], [13].

B. The Niederreiter cryptosystem and Classic McEliece

The Niederreiter cryptosystem [2] is the canonical example of a cryptosystem whose security relies on the hardness of the syndrome decoding problem. In this article, we focus on the encryption step, in particular on the syndrome computation, as shown on line 3 in Algorithm 1. Its implementation, as a matrix-vector multiplication, is detailed later.

During the encapsulation process of the *Classic McEliece* proposal [8], a matrix-vector multiplication between the public key and a uniform random vector $\boldsymbol{e}$ of low weight t is performed, just like on line 3 of Algorithm 1.

Algorithm 2 recalls the encapsulation process in *Classic McEliece*. It is straightforward to deduce that any adversary who can retrieve $\boldsymbol{e}$ from $\boldsymbol{s}^*$ and $\boldsymbol{H}_{\text{pub}}$ is able to break the Encapsulation function of the *Classic McEliece* KEM. Indeed, if one is able to find $\boldsymbol{e}$, this implies that one can compute $C = \mathsf{H}(2, \boldsymbol{e})$ (see line 4 in Algorithm 2) and thus find the secret key $K = \mathsf{H}(1 \parallel \boldsymbol{e} \parallel \boldsymbol{s}^* \parallel C)$ (see line 5 in Algorithm 2).

Algorithm 2 *Classic McEliece* encapsulation

```
1: function ENCAPS($\boldsymbol{H}_{\text{pub}}$)
2:     Generate a uniform random vector $\boldsymbol{e} \in \mathbb{F}_2^n$, $\text{HW}(\boldsymbol{e}) = t$
3:     Compute $\boldsymbol{s}^* \leftarrow \boldsymbol{H}_{\text{pub}}\boldsymbol{e}$
4:     Compute $C \leftarrow \mathsf{H}(2 \parallel \boldsymbol{e})$.        ▷ H is a hash function
5:     Compute $K \leftarrow \mathsf{H}(1 \parallel \boldsymbol{e} \parallel \boldsymbol{s}^* \parallel C)$
6:     return $(\boldsymbol{s}^* \parallel C)$ and $K$
```

TABLE I: *Classic McEliece* parameters

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

The sets of (n, k, t) parameters of the cryptosystem specified in the *Classic McEliece* proposal [8] are given in Table I. This is the set of parameters we consider in this article.

C. Embedded implementations

Before NIST started the post-quantum cryptography standardisation process, a number of embedded implementations of the Niederreiter cryptosystem had already been published [45], [17], [46]. Later on, as the *Classic McEliece* cryptosystem entered the NIST standardisation process, additional implementations were proposed as well. 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 [19], [20], [23]. Later on, more specific microcontroller implementations of the candidates were released [47]. However, the *Classic McEliece* cryptosystem was not included in this comparison since, as already noted in [22], 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 [21], [22].

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 3, 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 3, where an intermediate value b, which is a byte, is repeatedly XORed with the bitwise AND between one byte from the matrix row and one byte from the error vector. In [27], only the *schoolbook* version of the matrix-vector multiplication algorithm is attacked, where bits are stored individually in the machine words. The packed version remained out of reach, since attacking it required a multi-spot laser fault injection setup and a very large number of perfectly controlled faults. In the attack we propose here, we target the packed version, although the schoolbook version could be attacked in the same way.

Algorithm 3 Packed matrix-vector multiplication

```
1: function MAT_VEC_MULT_PACKED($\boldsymbol{H}$, $\boldsymbol{e}$)
2:     for $r \leftarrow 0$ to $((n-k)/8-1)$ do
3:         $\boldsymbol{s}_r \leftarrow \boldsymbol{0}$                                    ▷ Initialisation
4:     for $r \leftarrow 0$ to $(n-k-1)$ do
5:         $\boldsymbol{b} \leftarrow \boldsymbol{0}$
6:         for $c \leftarrow 0$ to $(n/8-1)$ do
7:             $\boldsymbol{b} \leftarrow \boldsymbol{b} \oplus (\boldsymbol{H}_{[r,c]} \wedge \boldsymbol{e}_c)$        ▷ Multipl. and addition
8:         $\boldsymbol{b} \leftarrow \boldsymbol{b} \oplus (\boldsymbol{b} >> 4)$                      ▷
9:         $\boldsymbol{b} \leftarrow \boldsymbol{b} \oplus (\boldsymbol{b} >> 2)$                      ▷ XOR folding
10:        $\boldsymbol{b} \leftarrow \boldsymbol{b} \oplus (\boldsymbol{b} >> 1)$                      ▷
11:        $\boldsymbol{b} \leftarrow \boldsymbol{b} \wedge 1$                              ▷ LSB extraction
12:        $\boldsymbol{s}^*_{\lfloor r/8 \rfloor} \leftarrow \boldsymbol{s}^*_{\lfloor r/8 \rfloor} \vee (\boldsymbol{b} << (r \bmod 8))$    ▷ Bit packing
13:    return $\boldsymbol{s}^*$
```

D. Physical attacks on the matrix-vector multiplication

In this section, we outline the attack proposed in [27], since the attack we propose follows the same framework. The first step consists in obtaining an integer syndrome instead of a binary syndrome. To this end, in [27], an XOR 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, one for each execution of the XOR instruction. A constant-time algorithm implementation does make the fault injection process easier in practice. However, 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 3. 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 if this is a standard attacker model for laser fault injection, it calls for simplification and a less restrictive attack path.

The second step of the attack consists in recovering the message from the integer syndrome. To this end, in [27], a computationally-efficient integer linear programming solver is used. Here, we propose to perform a dot product between the integer syndrome and the columns of the parity-check matrix to recover the positions of the errors in the error vector, as detailed in Section IV. Compared with the attack proposed in [27], the one we describe here is much more practical, since it is based on side-channel analysis only. It is also faster, and less sensitive to errors. A side-by-side comparison of the two attacks is provided in Table II.

III. Side-channel analysis to recover the integer syndrome

The first step of the attack proposed in [27] consists in obtaining an integer syndrome. In this section, we propose an alternative way to obtain it, that does not rely on fault injection, but on side-channel analysis only.

A. Side-channel measurements

We perform side-channel measurements during the syndrome computation. 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 $\text{HW}(\boldsymbol{e}) = t$. 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 3). They will be used as labels for the classifier.

B. 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.

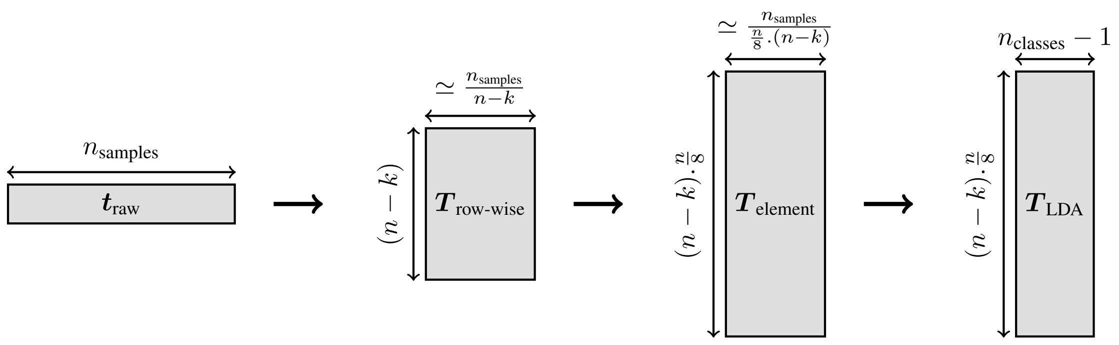
1) First reshaping : matrix-row level: The first preprocessing step consists in reshaping the vector $\boldsymbol{t}_{\text{raw}}$ into a matrix, with each row corresponding to the multiplication of one row of $\boldsymbol{H}$ with the error vector $\boldsymbol{e}$. This is easily done since the matrix-vector multiplication is a very regular operation, in which patterns can be identified by visual inspection. These patterns correspond to iterations of the outer `for` loop, on line 4 in Algorithm 3.

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 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 $\boldsymbol{e}$.

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 $\boldsymbol{e}$. Again this is easily done since the vector-vector multiplication is a very regular operation, in which patterns can be identified easily. These patterns correspond to iterations of the inner `for` loop, on line 6 in Algorithm 3.

TABLE II: Comparison of the fault injection attack from [27] and the proposed side-channel attack

Criterion	**Fault injection attack from [27]**	**Proposed side-channel attack**
Method to obtain a binary syndrome	instruction corruption by laser fault injection	side-channel analysis
Derivation of the exact value of the binary syndrome	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 binary syndrome	integer linear programming	dot-product and information-set decoding
Practicality	expensive setup and restrictive attacker model	affordable setup and loose attacker model



n_{samples}
$\boldsymbol{t}_{\text{raw}}$
$\simeq \frac{n_{\text{samples}}}{n-k}$
$(n-k)$
$\boldsymbol{T}_{\text{row-wise}}$
$\simeq \frac{n_{\text{samples}}}{\frac{n}{8}.(n-k)}$
$(n-k).\frac{n}{8}$
$\boldsymbol{T}_{\text{element}}$
$n_{\text{classes}} - 1$
$(n-k).\frac{n}{8}$
$\boldsymbol{T}_{\text{LDA}}$

Fig. 1: Summary of the preprocessing steps applied to the raw traces

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 finished. In fact, these extra samples correspond to the XOR folding operation, the LSB extraction and the bit-level storage of the syndrome (see lines 8 to 12 in Algorithm 3).

3) Linear Discriminant Analysis: After those two reshaping steps, each row of the matrix corresponds to the multiplication of one byte of the matrix $\boldsymbol{H}$ with one byte 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 [48], 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. Therefore, the $\boldsymbol{T}_{\text{LDA}}$ matrix has eight columns.

4) Summary: Figure 1 summarises the preprocessing steps which are applied to the raw side-channel measurements, showing the changes of dimensions at every step.

These preprocessing steps actually have two considerable 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 matrix-vector multiplication 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 n changes, only *one* classifier must be trained. This is very interesting for the training complexity, 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 $(n-k)\frac{n}{8} = 1664 \times 1024 \approx 1.7 \times 10^6$ training samples. Consequently, provided that an attacker can record a sufficiently large number of samples, a single side-channel trace is sufficient to build the training set.

C. 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 [49].

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. This allows random forests to achieve high accuracy, while being easy to train. An additional feature of random forests is called "bagging": each tree is trained on a subset of the training samples only, and considering a subset of the features only. This makes random forest less sensitive to overfitting on the training data. We refer the reader to [50] 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 $\boldsymbol{t}_{\text{test}}$.

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)).

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

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 $\boldsymbol{T}_{\text{LDA}}$.

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

The Δ value is empirically selected. Experiments presented in Section V-C 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.

D. Derivation of the integer syndrome

Once the consecutive Hamming weights of the intermediate value are recovered, an approximate value for the integer syndrome entries may be derived. Let $\widetilde{\boldsymbol{s}}$ denote this integer vector. When $\widetilde{\boldsymbol{s}} = \boldsymbol{s}$ our approximation is perfect. 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 3). Computing the value of the integer syndrome entry 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).

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

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 V. Therefore, we propose the following formula to derive the value of the integer syndrome entries from the Hamming weights, given in Equation (3) and detailed in Lemma 1.

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

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 $\text{HW}(\boldsymbol{a}) \ge \text{HW}(\boldsymbol{b})$. Let $\overline{\boldsymbol{a}} = \boldsymbol{1}_\ell - \boldsymbol{a}$ be the bitwise complement of $\boldsymbol{a}$. Let $\boldsymbol{a}\&\boldsymbol{b} = (a_1b_1, \ldots, a_\ell b_\ell)$ be the element-wise product of the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. Then we have:

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

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

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

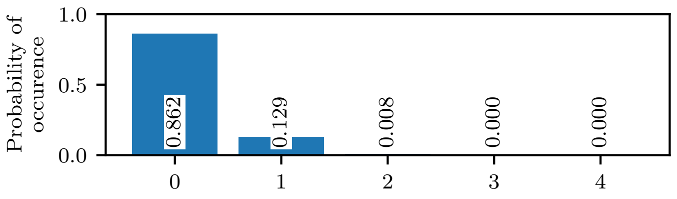
Notice that a particular case where the aforementioned condition $\mathsf{Supp}(\boldsymbol{b}) \subseteq \mathsf{Supp}(\boldsymbol{a})$ is valid is when $\text{HD}(\boldsymbol{a}, \boldsymbol{b}) \le 1$, as specified in Equation (3).

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.

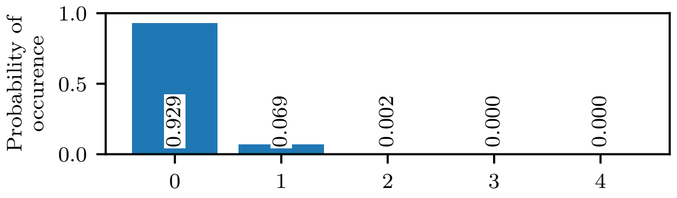
However, we are not in this general case here, since the error vector is of low Hamming weight (see the *Classic McEliece* parameters in Table I). For example, for the lowest value of $n = 3488$, the error vector has a Hamming weight of 64 only. For the highest value of $n = 8192$, the error vector has 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 XORed with the previous value of $\boldsymbol{b}$ (see line 7 in Algorithm 3), 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 $\frac{\frac{n}{8}-t}{\frac{n}{8}} = \frac{436-64}{436} = \frac{372}{436} = 85.32\,\%$.



(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.

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

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.

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 XORed 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 integer syndrome.

It is important to remark from Eq. (4) that $\widetilde{s}_i \leq s_i$. 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 V.

E. *Extra parity-correction step*

After computing $\widetilde{\boldsymbol{s}}$ an additional parity correction step can be followed. The information about the parity of the integer syndrome can be deduced from $\boldsymbol{s}^*$. Since $\widetilde{\boldsymbol{s}} \leq \boldsymbol{s}$ (component-wise) one can add the parity value to the estimated value.

For example, if we estimate the value of an integer syndrome entry to be 14, but the associated binary syndrome entry is 1, then we know that the value 14 is wrong, since it is even but the binary syndrome entry is odd. The estimated integer syndrome entry is then changed to 15.

What we eventually obtain is a better $\widetilde{\boldsymbol{s}}$. More precisely $\boldsymbol{s}^*$ is the correct binary syndrome but the entries of $\boldsymbol{s}$ might be incorrect, although with the correct parity. Even if we might have $\widetilde{\boldsymbol{s}} \neq \boldsymbol{s}$ (component-wise), the next section shows how to exploit it to recover the message.

IV. Message recovery algorithms

The second step of the attack proposed in [27] consists 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 integer syndrome leads to an incorrect result. Thus the integer syndrome must be perfectly correct. Second, the complexity of those algorithms was not systematically analysed in [27].

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 stress out that the algorithm proposed here is independent of the first step. More exactly, regardless of the technique employed to recover an integer syndrome, the solution proposed here works as long as the recovered syndrome is "close" enough to the integer syndrome. Let us start by recalling the definition of the $\mathbb{N}-\mathsf{SDP}$.

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 $\boldsymbol{He} = \boldsymbol{s}$.

A. *Identification of a "good" permutation*

The main idea behind the algorithms proposed here is to use techniques such as those in [33] to find a "good" initial permutation for the information-set decoder.

Compared with classical approaches in information-set decoding, where permutations are exhaustively enumerated, we exploit the extra information coming from the integer syndrome to 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 integer syndrome. 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 integer syndrome, the scalar product should find which columns point in the same direction as the integer syndrome.

To increase the correlation between columns in the support of the error vector and the syndrome, 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 can now define the score function.

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:

$$\forall i \in \mathbb{N}^*_n \quad \psi_i(\boldsymbol{s}) = \boldsymbol{H}_{[,i]} \cdot \boldsymbol{s} + \overline{\boldsymbol{H}}_{[,i]} \cdot \overline{\boldsymbol{s}} \tag{6}$$

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, as detailed in Algorithm 4.

The score function, as pointed in [33], has the ability of discriminating between columns belonging to the support of $\boldsymbol{e}$, where the t errors lie, and the rest of the columns of $\boldsymbol{H}$, as long as $n-k$ grows significantly compared to t. The theoretical argument resides in the distribution of $\psi_i(\boldsymbol{s})$.

Typically, one has for $i \notin \mathsf{Supp}(\boldsymbol{e})$ an expected value of $\psi_i(\boldsymbol{s})$ around $\frac{(n-k).t}{2}$, while for $i \in \mathsf{Supp}(\boldsymbol{e})$ we have $\psi_i(\boldsymbol{s})$ concentrated around $\frac{n-k}{2} + \frac{(n-k).t}{2}$. Hence, there is a gap between the average values equal to $\frac{n-k}{2}$, 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 with respect to $(n-k)$. This fact suggests that the probability of discriminating indices in the support of the error vector increases when $(n-k)$ increases.

In [33] a lower bound on the probability of success of score-based algorithms is given (t-Threshold Score Decoder and Rank-Threshold Score Decoder). However, their analyses are only valid for large values of parameters. The *Classic McEliece* parameters are too small with respect to the theoretical analysis. Therefore, we will mainly rely here on simulations to estimate the success of the score-based decoders.

The integer syndrome, which might be partially incorrect as detailed in Section III-D is only used up to this point, to identify a good permutation. If we denote the recovered syndrome by $\widetilde{s} = s + \varepsilon$, where ε denotes the random variable describing the faults, the sort function will take as input $\widetilde{s}$ instead of $\boldsymbol{s}$. In this case, the theoretical analysis from [33] does not hold any more.

Our simulations show that under certain conditions, the value $\psi_i(\widetilde{\boldsymbol{s}})$ deviates from the correct value $\psi_i(\boldsymbol{s})$ in an identical manner for positions $i \in \mathsf{Supp}(\boldsymbol{e})$ and $i \notin \mathsf{Supp}(\boldsymbol{e})$. Quantifying or characterizing the type of errors tolerated by ψ represents a complete and independent research direction. We only carry it out from an experimental point of view here.

Moving forward, there are two direct algorithms for retrieving the error positions using the permutation $\boldsymbol{\Pi}$ returned by $\textsc{sort}(\boldsymbol{H}, \widetilde{\boldsymbol{s}}, t)$. Hence, from now on, only the ciphertext, *i.e.* the binary syndrome, is used.

Algorithm 4 Sort function

```
1: function SORT($\boldsymbol{H}, \boldsymbol{s}, t$)
2:     for $i \leftarrow 1$ to $n$ do
3:         compute $\psi_i(\boldsymbol{s})$ with Equation (6)
4:     $\boldsymbol{\Pi} \leftarrow$ sort $\psi_i(\boldsymbol{s})$ in descending order
5:     return $\boldsymbol{\Pi}$
```

B. ISD-based algorithms that exploit the good permutation

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 $\boldsymbol{e} = \boldsymbol{\Pi}(\boldsymbol{1}_t \parallel \boldsymbol{0}_{n-t})^T$. 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 4 is $\mathcal{O}((n-k)n) = \mathcal{O}(n^2)$. Equation (6) implies that the overall time complexity of Algorithm 4 reduces to three matrix-vector multiplications, *i.e.* $\widetilde{\boldsymbol{s}}^T\boldsymbol{H}$, $\overline{\widetilde{\boldsymbol{s}}}^T\overline{\boldsymbol{H}}$ and $\boldsymbol{H\Pi}(\boldsymbol{1}_t \parallel \boldsymbol{0}_{n-t})^T$, and sorting a list of n integers.

Proposition 1. `t-Threshold Score Decoder` outputs a valid solution as long as $\min\{\psi_i(\widetilde{\boldsymbol{s}}), i \in \mathsf{Supp}(\boldsymbol{e})\} > \max\{\psi_i(\widetilde{\boldsymbol{s}}), i \in \mathbb{N}^*_n \setminus \mathsf{Supp}(\boldsymbol{e})\}$.

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

Experiments show 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.

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 $\textsc{sort}(\boldsymbol{H}, \boldsymbol{s}, t)$ function to obtain a permutation $\boldsymbol{\Pi}$ for $\boldsymbol{H}$. Then, use the binary syndrome $\boldsymbol{s}^*, \boldsymbol{\Pi}$ and $\boldsymbol{H}$ to retrieve $\boldsymbol{e}$ by means of linear algebra. We start by computing the reduced row-echelon form $\boldsymbol{A}^*, \boldsymbol{H}^* \leftarrow \texttt{rref}(\boldsymbol{H\Pi})$. If $\mathrm{HW}(\boldsymbol{A}^*\boldsymbol{s}^*)$ 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, $\boldsymbol{A}^*\boldsymbol{H\Pi} = \begin{bmatrix} \boldsymbol{I_r} \\ \boldsymbol{0_{n-k-r,r}} \end{bmatrix} \parallel \boldsymbol{B}^* \Bigg]$ where $\boldsymbol{H\Pi} = [\boldsymbol{A} \parallel \boldsymbol{B}]$ with $\boldsymbol{A}$

Algorithm 5 Lee-Brickell Score Decoder

```
1: function LEE-BRICKELL SCORE DECODER($\boldsymbol{H}, \boldsymbol{s}^*, t$)
2:     Compute $\boldsymbol{\Pi} \leftarrow$ SORT($\boldsymbol{H}, \widetilde{\boldsymbol{s}}, t$)
3:     Set $\boldsymbol{H\Pi} \leftarrow [\boldsymbol{A} \parallel \boldsymbol{B}]$
4:     Compute $\boldsymbol{A}^*, \boldsymbol{H}^* \leftarrow$rref($\boldsymbol{H\Pi}$) and $\boldsymbol{B}^* \leftarrow \boldsymbol{A}^*\boldsymbol{B}$
5:     Compute $\boldsymbol{s}^{'} \leftarrow \boldsymbol{A}^*\boldsymbol{s}^*$
6:     if HW($\boldsymbol{s}^{'}$) $== t$ then
7:         return $\boldsymbol{e} \leftarrow \boldsymbol{\Pi}(\boldsymbol{s}^{'} \parallel \boldsymbol{0}_k)^T$
8:     else
9:         for $i \leftarrow 1$ to $\delta$ do
10:            S $\leftarrow$ Gener-Subsets($\{1,\dots,n-r\}$,$i$)
11:            for $E$ in S do
12:                $\boldsymbol{e}^{''} \leftarrow$ Vector($\{0,1\}, n-r, E$)
13:                $\boldsymbol{e}^{'} \leftarrow \boldsymbol{s}^{'} - \boldsymbol{B}^*\boldsymbol{e}^{''}$
14:                if HW($\boldsymbol{e}^{'}$) $== t - i$ then
15:                    return $\left(\boldsymbol{\Pi}(\boldsymbol{e}^{'} \parallel \boldsymbol{e}^{''})^T, \boldsymbol{\Pi}\right)$
```

an $(n-k) \times r$ matrix satisfying $\boldsymbol{A}^*\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I_r} \\ \boldsymbol{0_{n-k-r,r}} \end{bmatrix}$, and $\boldsymbol{B}^* = \boldsymbol{A}^*\boldsymbol{B}$.

Proposition 2. 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 $\min\{\psi_i(\widetilde{\boldsymbol{s}}), i \in \mathsf{Supp}(\boldsymbol{e})\} > \max\{\psi_i(\widetilde{\boldsymbol{s}}), i \in L\}$.

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 $\boldsymbol{A}^*$.

3) Lee-Brickell Score Decoder: Rank-Threshold Score Decoder uses linear algebra in the same way as the Prange decoder does for syndrome decoding [38]. 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 5.

Remark 3. The working factor of Algorithm 5 is given by

$$O((n-k)^3) + O\left(\binom{n-r}{\delta}(n-r)^2\right), \qquad (7)$$

where the first term corresponds to the work factor of rref($\boldsymbol{H\Pi}$), 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 time complexity of $\mathcal{O}(n^4)$.

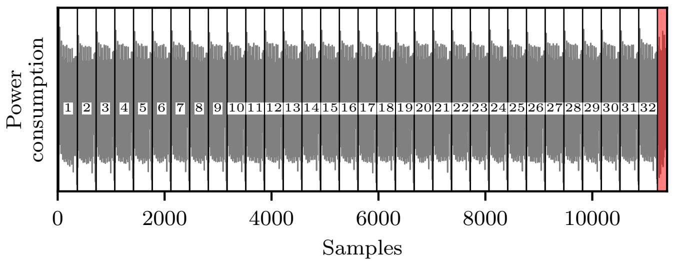
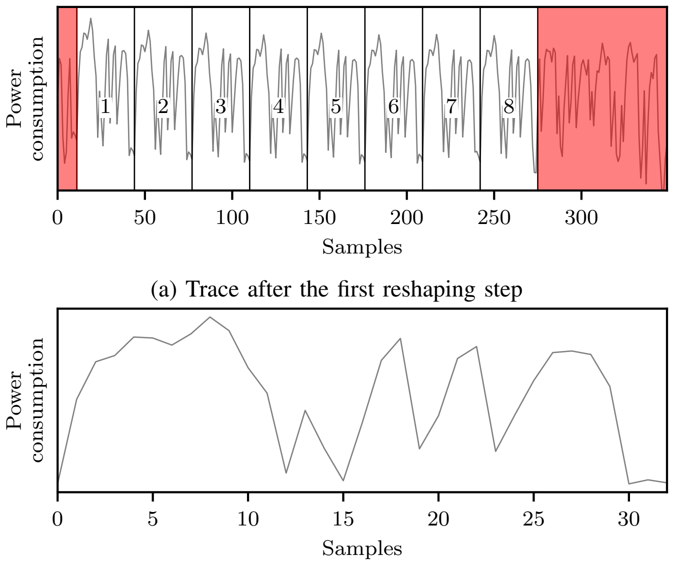


Fig. 3: Side-channel trace of the syndrome computation ($n = 64$ for readability)



(a) Trace after the first reshaping step

(b) Trace after the second reshaping step

Fig. 4: Side-channel traces obtained after the first and the second reshaping steps.

V. Experimental results

A. Side-channel measurements

All the side-channel measurements are power consumption measurements. They were performed using the ChipWhisperer platform [51]. The targets microcontroller that computes the syndrome is based on an ARM® Cortex®-M4 core, which is the hardware platform recommended by NIST, and already used in related work such as the pqm4 library [47] to benchmark and test implementations of post-quantum cryptography algorithms. We use the reference implementation of the matrix-vector multiplication algorithm provided in the *Classic McEliece* submission [8]. Figure 3 shows a side-channel trace measured during the syndrome computation.

For readability reasons, very small parameters are used for the plots in Figures 3 and 4, with $n = 64$, $k = 32$ and $t = 8$, but the reasoning applies in the same way for larger 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 V-B. For the rest of the experiments, the values of n, k and t we considered are given in Table I.

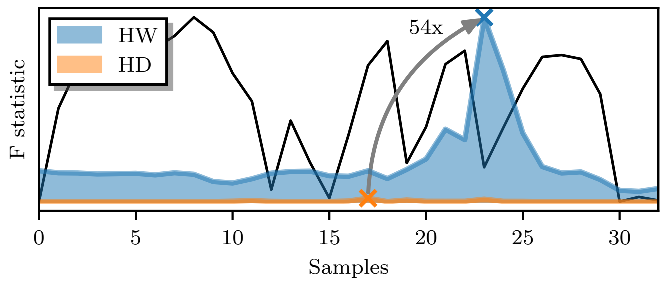


Fig. 5: Signal-to-noise ratio for Hamming weight (HW) and Hamming distance (HD) leakages, superimposed with the power consumption data from Figure 4b.

B. Side-channel traces preprocessing

The raw trace from Figure 3 is reshaped one time to obtain the trace shown in Figure 4a, which is one of the $(n-k) = 32$ patterns labelled in Figure 3. This corresponds to the multiplication of one row of the matrix $\boldsymbol{H}$ with the error vector $\boldsymbol{e}$. This reshaping is easily done by observing the trace from Figure 3 and finding the width of a pattern, which is constant.

Then, the trace from Figure 4a is reshaped a second time to obtain the trace shown in Figure 4b, which is one of the eight patterns labelled 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 in Algorithm 3: the exclusive-OR folding, the LSB extraction and the bit packing operations.

As claimed above in Section III, 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 preprocessed trace. The signal-to-noise ratio is computed by an F-test, *i.e.* 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.

The last preprocessing step consists in reducing the dimension of the data to make it easier to handle by the classifier. We used `LinearDiscriminantAnalysis`[1] from the `sklearn` Python package to map the preprocessed traces 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, traces are fed to the classifier for training and inference.

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

TABLE III: Training samples and classifier accuracy

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 (σ)	0.9964 (0.0021)	0.9983 (0.00092)	0.9963 (0.00076)	0.9983 (0.00039)

C. Hamming weight recovery with a random forest

We used `RandomForestClassifier`[2], from the `sklearn` Python package, as an implementation of a random forest classifier, sticking with the default parameters. In particular, there are one hundred 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. Ten classifiers were then trained and their accuracy computed from the test set. The accuracy was always very high, ranging from 99.17 % to 99.91 %, with $\Delta = 3$, where Δ is the concatenation parameter discussed in Section III-C. With $\Delta < 3$, the accuracy drops, while with $\Delta > 3$, the dimensionality reduction step exceeds the memory capacity of the computer we use, which is 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 III.

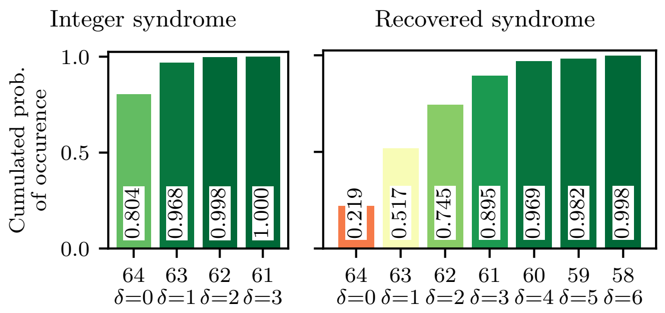
D. Derivation of the integer syndrome

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. A significant source of errors in this step is when the Hamming weight of the intermediate value remains the same, but two bits actually flipped in opposite directions. This happens for example when the intermediate value b changes from `0b00000001` to `0b00000010`. The Hamming distance is two in this case. However, is considered to be zero when looking at the Hamming weights only. This has been discussed before and shown in Figure 2b. Even though this happens very rarely, this is still visible as a result of the summation over large n values when applying Equation (3). As a result, the integer syndrome is never perfectly recovered by this method. We shall now see how this affects the performance of the score function.

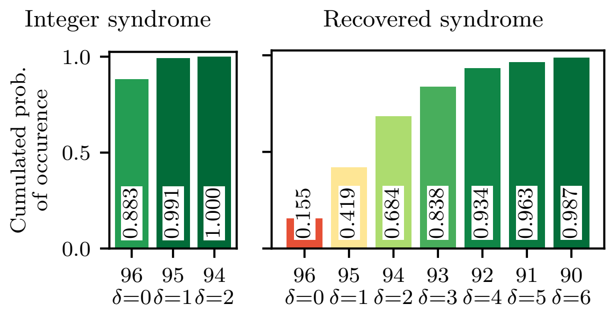
E. Evaluation of the performance of the score function

After recovering the integer syndrome, we now assess the performance of the score function, *i.e.* its ability to find the er-

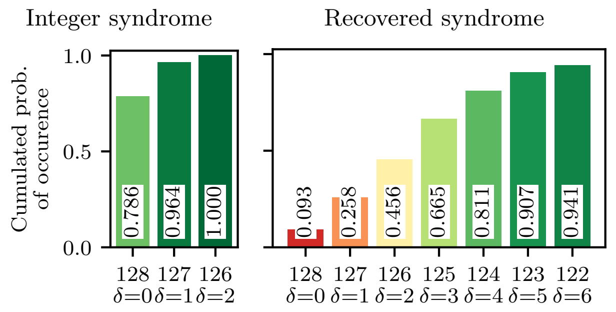
[2]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html



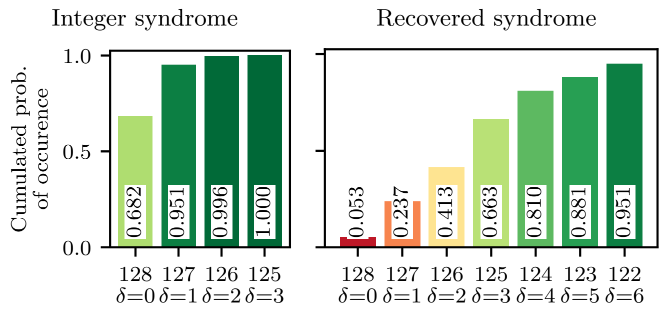
(a) $n = 3488$, $k = 2720$, $t = 64$



(b) $n = 4608$, $k = 3360$, $t = 96$



(c) $n = 6688$, $k = 5024$, $t = 128$



(d) $n = 8192$, $k = 6528$, $t = 128$

Fig. 6: Probability of finding at least x ones in the first $(n-k)$ positions for the integer and recovered syndrome

ror positions in the error vector. The first set of simulations are done using the integer syndromes. Given a public parity-check matrix $\boldsymbol{H}$, 10 000 random error vectors are generated and the associated integer syndromes are computed as a matrix-vector product. For each integer syndrome, the probability of success of the LEE-BRICKELL SCORE DECODER for all sets of parameters with respect to the value of δ is computed. We recall that the LEE-BRICKELL SCORE DECODER succeeds if $t-\delta$ ones are in the first $n-k$ positions in the error vector. Results are shown in Figure 6, under the "Integer syndrome" plots. Notice that, in this case, $\delta = 2$ gives a probability of success greater than or equal to 0.995 for all the parameters. In order to increase it up beyond 0.999, taking $\delta = 3$ is sufficient.

For the second set of simulations, the integer syndromes are recovered by exploiting the simulated Hamming weight information. This is fed to Equation (3) and the parity-correction step. This allows to estimate the performance of the score function more accurately, when the integer syndrome is recovered from side-channel analysis. Here, we perform again 10 000 simulations for each parameters set. Results are shown in Figure 6, under the "Recovered syndrome" plots. We observe that, this time, a larger δ value is necessary to reach the same probability of success compared with the case where the integer syndrome is used. The reason for this is that the integer syndrome is not perfectly recovered, as detailed above. Still, overall, a high probability of success is achieved for values of δ up to 6. Beyond this value, the computational complexity of the information-set decoding process becomes prohibitive.

VI. COUNTERMEASURES

The weakness of cryptosystems based on the syndrome decoding problem that makes it vulnerable to the proposed attack is that the error weight t is small. Increasing t cannot be done, since it depends on the correction capacity of the underlying code. Instead, masking can be used to produce an error vector with more errors.

A. *Classical masking*

The leakage used to obtain the integer syndrome can be hidden with a mask. Indeed, using vectors of random Hamming weight $\frac{n}{2}$ limits the accuracy of the random forest method to recover the Hamming weights. Indeed, the concatenation technique that considers adjacent matrix rows is not applicable anymore, since the Hamming weight variations are large. As a consequence, the recovered integer syndrome is "more incorrect". Therefore, the permutation $\boldsymbol{\Pi} \leftarrow \text{SORT}(\boldsymbol{H}, \boldsymbol{s}, t)$ is not suitable and we must resort to exhaustive enumeration in information-set decoding, which is computationally expensive.

The proposed masking scheme is as follows. We first draw a random value $\boldsymbol{x} \in \mathbb{F}_2^{n-k}$ and add it to $\boldsymbol{e}$. Then, two matrix-vector products are performed: $\boldsymbol{H}_{\text{pub}}(\boldsymbol{e} \oplus \boldsymbol{x})$ and $\boldsymbol{H}_{\text{pub}}\boldsymbol{x}$. We have $\boldsymbol{H}_{\text{pub}}\boldsymbol{e} = \boldsymbol{H}_{\text{pub}}(\boldsymbol{e} \oplus \boldsymbol{x}) \oplus (\boldsymbol{H}_{\text{pub}}\boldsymbol{x})$ since the operations are linear. This is shown in Algorithm 6.

The overhead of such a masked implementation is linear in both time and randomness. Both matrix-vector products are performed with vectors of average Hamming weight equal to $\frac{n}{2}$. Performing the attack twice, once on $\boldsymbol{H}_{\text{pub}}(\boldsymbol{e} \oplus \boldsymbol{x})$ and once on $\boldsymbol{H}_{\text{pub}}\boldsymbol{x}$, and combining the results, is unlikely to work since the average Hamming weight is now $\frac{n}{2}$ instead of $t \leq \frac{n-k}{\log_2 n} << \frac{n}{2}$, as dictated by the decoding capacity

Algorithm 6 Encryption protected with classical masking

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

of binary Goppa codes. Another way to attack is to directly perform a second-order attack. However, in that case, we know that masking theoretically increases the difficulty of an attack exponentially in the number of shares [52]. Therefore, the attack remains difficult in this case too.

B. 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 7, where the mask is computed as $\boldsymbol{x}\,\boldsymbol{G}_{\text{pub}}$.

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 $\boldsymbol{x}\,\boldsymbol{G}_{\text{pub}}$ can be seen as a random bit. There exists 2^k different masks $\boldsymbol{x}\,\boldsymbol{G}_{\text{pub}}$. 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.

VII. Conclusion

This article presents a message recovery attack against the matrix-vector multiplication algorithm, which is commonly used in cryptosystems whose security relies on the hardness of the syndrome decoding problem. The attack uses side-channel analysis on the power consumption and machine learning techniques. The attack works in two steps. First, we retrieve the value of the integer syndrome, 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 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.

Preliminary simulations indicate that the parameters for *Classic McEliece*, even though smaller than those for BIKE in terms of code-length and dimension, are more robust against Score based decoders. More exactly, the `Rank-Threshold Score Decoder` requires a smaller percentage of syndrome entries when BIKE parameters are used compared to *Classic McEliece* parameters. The main argument resides in smaller values of $t/(n-k)$ for BIKE compared with *Classic McEliece*.

Algorithm 7 Encryption protected with a random codeword

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

Therefore, we believe that the proposed attack has a high potential of application in the context of other code-based solutions, *e.g.* BIKE.

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 is very relevant.

Another research direction could be to further improve the derivation of the integer syndrome. Indeed, while we considered only the Hamming weight information in this work, the Hamming distance could be exploited as well, in order to obtain a more precise recovery. Further studies could focus on combining those two leakages in an efficient manner.

Finally, although we suggested 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.

Brice Colombier 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”.

Authors are particularly grateful to the referees for their constructive inputs and comments.

References

[1] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg, “On the inherent intractability of certain coding problems (corresp.),” *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[2] H. Niederreiter, “Knapsack-type cryptosystems and algebraic coding theory,” *Problems of Control and Information Theory*, vol. 15, no. 2, pp. 159–166, 1986.

[3] D. Augot, M. Finiasz, and N. Sendrier, “A family of fast syndrome based cryptographic hash functions,” in *First International Conference on Cryptology in Malaysia*, ser. Lecture Notes in Computer Science, E. Dawson and S. Vaudenay, Eds., vol. 3715. Kuala Lumpur, Malaysia: Springer, Sep. 2005, pp. 64–83.

[4] P. Gaborit, C. Lauradoux, and N. Sendrier, “SYND: a fast code-based stream cipher with a security reduction,” in *International Symposium on Information Theory*. Nice, France: IEEE, Jun. 2007, pp. 186–190.

[5] J. Stern, “A new identification scheme based on syndrome decoding,” in *Annual International Cryptology Conference*, ser. Lecture Notes in Computer Science, D. R. Stinson, Ed., vol. 773. Santa Barbara, California, USA: Springer, Aug. 1993, pp. 13–21.

[6] J. Fischer and J. Stern, "An efficient pseudo-random generator provably as secure as syndrome decoding," in *International Conference on the Theory and Application of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, U. M. Maurer, Ed., vol. 1070. Saragossa, Spain: Springer, May 1996, pp. 245–255.

[7] N. Aragon, P. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Guneysu, C. Aguilar Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, G. Zémor, V. Vasseur, and S. Ghosh, "BIKE," National Institute of Standards and Technology, Tech. Rep., 2020.

[8] M. R. Albrecht, D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. Lange, V. Maram, I. von Maurich, R. Misoczki, R. Niederhagen, K. G. Paterson, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, C. J. Tjhai, M. Tomlinson, and W. Wang, "Classic McEliece," National Institute of Standards and Technology, Tech. Rep., 2020.

[9] R. J. McEliece, "A public-key cryptosystem based on algebraic coding theory," Jet Propulsion Laboratory, California Institute of Technology, The Deep Space Network Progress Report 42-44, Jan./Feb. 1978.

[10] A. May, A. Meurer, and E. Thomae, "Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$," in *International Conference on the Theory and Application of Cryptology and Information Security*, ser. Lecture Notes in Computer Science, D. H. Lee and X. Wang, Eds., vol. 7073. Seoul, South Korea: Springer, Dec. 2011, pp. 107–124.

[11] A. Becker, A. Joux, A. May, and A. Meurer, "Decoding random binary linear codes in $2^{n/20}$: How 1 + 1 = 0 improves information set decoding," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, D. Pointcheval and T. Johansson, Eds., vol. 7237. Cambridge, UK: Springer, Apr. 2012, pp. 520–536.

[12] A. May and I. Ozerov, "On computing nearest neighbors with applications to decoding of binary linear codes," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9056. Sofia, Bulgaria: Springer, Apr. 2015, pp. 203–228.

[13] L. Both and A. May, "Decoding linear codes with high error rate and its impact for LPN security," in *International Conference on Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science, T. Lange and R. Steinwandt, Eds., vol. 10786. Fort Lauderdale, FL, USA: Springer, Apr. 2018, pp. 25–46.

[14] A. Esser, A. May, and F. Zweydinger, "McEliece needs a break - solving McEliece-1284 and Quasi-Cyclic-2918 with modern ISD," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, O. Dunkelman and S. Dziembowski, Eds., vol. 13277. Trondheim, Norway: Springer, May 2022, pp. 433–457.

[15] S. Heyse, "Low-Reiter: Niederreiter encryption scheme for embedded microcontrollers," in *International Workshop on Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science, N. Sendrier, Ed., vol. 6061. Darmstadt, Germany: Springer, May 2010, pp. 165–181.

[16] S. Heyse and T. Güneysu, "Code-based cryptography on reconfigurable hardware: tweaking Niederreiter encryption for performance," *Journal of Cryptographic Engineering*, vol. 3, no. 1, pp. 29–43, 2013.

[17] I. von Maurich, L. Heberle, and T. Güneysu, "IND-CCA secure hybrid encryption from QC-MDPC Niederreiter," in *International Workshop on Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science, T. Takagi, Ed., vol. 9606. Fukuoka, Japan: Springer, Feb. 2016, pp. 1–17.

[18] W. Wang, J. Szefer, and R. Niederhagen, "FPGA-based Niederreiter cryptosystem using binary Goppa codes," in *International Conference on Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science, T. Lange and R. Steinwandt, Eds., vol. 10786. Fort Lauderdale, FL, USA: Springer, Apr. 2018, pp. 77–98.

[19] K. Basu, D. Soni, M. Nabeel, and R. Karri, "NIST post-quantum cryptography- A hardware evaluation study," *IACR Cryptology ePrint Archive*, p. 47, 2019.

[20] V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, and K. Gaj, "Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches," *IACR Cryptology ePrint Archive*, p. 795, 2020.

[21] J. Roth, E. G. Karatsiolis, and J. Krämer, "Classic McEliece implementation with low memory footprint," in *International Conference on Smart Card Research and Advanced Applications*, ser. Lecture Notes in Computer Science, P. Liardet and N. Mentens, Eds., vol. 12609. Virtual Event: Springer, Nov. 2020, pp. 34–49.

[22] M. Chen and T. Chou, "Classic McEliece on the ARM Cortex-M4," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 3, pp. 125–148, 2021.

[23] V. Kostalabros, J. Ribes-González, O. Farràs, M. Moretó, and C. Hernández, "HLS-based HW/SW co-design of the post-quantum Classic McEliece cryptosystem," in *International Conference on Field-Programmable Logic and Applications*. Dresden, Germany: IEEE, Aug. 2021, pp. 52–59.

[24] N. Lahr, R. Niederhagen, R. Petri, and S. Samardjiska, "Side channel information set decoding using iterative chunking - plaintext recovery from the "Classic McEliece" hardware reference implementation," in *International Conference on the Theory and Application of Cryptology and Information Security*, ser. Lecture Notes in Computer Science, S. Moriai and H. Wang, Eds., vol. 12491. Daejeon, South Korea: Springer, Dec. 2020, pp. 881–910.

[25] J. Danner and M. Kreuzer, "A fault attack on the Niederreiter cryptosystem using binary irreducible Goppa codes," *Journal of Groups, Complexity, Cryptology*, vol. 12, no. 1, pp. 1–20, 2020.

[26] K. Xagawa, A. Ito, R. Ueno, J. Takahashi, and N. Homma, "Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates," in *International Conference on the Theory and Application of Cryptology and Information Security*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang, Eds., vol. 13091. Singapore: Springer, Dec. 2021, pp. 33–61.

[27] P. Cayrel, B. Colombier, V. Dragoi, A. Menu, and L. Bossuet, "Message-recovery laser fault injection attack on the Classic McEliece cryptosystem," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, A. Canteaut and F. Standaert, Eds., vol. 12697. Zagreb, Croatia: Springer, Oct. 2021, pp. 438–467.

[28] A. Horlemann, S. Puchinger, J. Renner, T. Schamberger, and A. Wachter-Zeh, "Information-set decoding with hints," in *International Workshop on Code-Based Cryptography*, ser. Lecture Notes in Computer Science, A. Wachter-Zeh, H. Bartz, and G. Liva, Eds., vol. 13150. Munich, Germany: Springer, Jun. 2021, pp. 60–83.

[29] V.-F. Dragoi, P.-L. Cayrel, B. Colombier, D. Bucerzan, and S. Hoara, "Solving a modified syndrome decoding problem using integer programming," *International Journal of Computers Communications & Control*, vol. 15, no. 5, 10 2020.

[30] R. Dorfman, "The detection of defective members of large populations," *Annals of Mathematical Statistics*, vol. 14, pp. 436–440, 1943.

[31] K. Lee, K. Chandrasekher, R. Pedarsani, and K. Ramchandran, "SAFFRON: A fast, efficient, and robust framework for group testing based on sparse-graph codes," *IEEE Transactions on Signal Processing*, vol. 67, no. 17, pp. 4649–4664, 2019.

[32] O. Gebhard, M. Hahn-Klimroth, D. Kaaser, and P. Loick, "Quantitative group testing in the sublinear regime," *CoRR*, vol. abs/1905.01458, 2019.

[33] U. Feige and A. Lellouche, "Quantitative group testing and the rank of random matrices," *CoRR*, vol. abs/2006.09074, 2020.

[34] R. C. Torres and N. Sendrier, "Analysis of information set decoding for a sub-linear error weight," in *International Workshop on Post-Quantum Cryptography*, ser. Lecture Notes in Computer Science, T. Takagi, Ed., vol. 9606. Fukuoka, Japan: Springer, Feb. 2016, pp. 144–161.

[35] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "A finite regime analysis of information set decoding algorithms," *Algorithms*, vol. 12, no. 10, p. 209, 2019.

[36] E. Kirshanova and T. Laarhoven, "Lower bounds on lattice sieving and information set decoding," in *Annual International Cryptology Conference*, ser. Lecture Notes in Computer Science, T. Malkin and C. Peikert, Eds., vol. 12826. Virtual Event: Springer, Aug. 2021, pp. 791–820.

[37] A. Esser and E. Bellini, "Syndrome decoding estimator," in *IACR International Conference on Practice and Theory of Public-Key Cryptography*, ser. Lecture Notes in Computer Science, G. Hanaoka, J. Shikata, and Y. Watanabe, Eds., vol. 13177. Virtual Event: Springer, Mar. 2022, pp. 112–141.

[38] E. Prange, "The use of information sets in decoding cyclic codes," *IRE Transactions on Information Theory*, vol. 8, no. 5, pp. 5–9, 1962.

[39] J. Stern, "A method for finding codewords of small weight," in *Proceedings of the 3rd International Colloquium on Coding Theory and Applications*. Berlin, Heidelberg: Springer-Verlag, Nov 1988, p. 106–113.

[40] P. J. Lee and E. F. Brickell, "An observation on the security of mceliece's public-key cryptosystem," in *Workshop on the Theory and Application of of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, C. G. Günther, Ed., vol. 330. Davos, Switzerland: Springer, May 1988, pp. 275–280.

[41] J. S. Leon, “A probabilistic algorithm for computing minimum weights of large error-correcting codes,” *IEEE Transactions on Information Theory*, vol. 34, no. 5, pp. 1354–1359, 1988.

[42] I. Dumer, “Two decoding algorithms for linear codes,” *Problems of Information Transmission*, vol. 25, no. 1, pp. 17–23, 1989.

[43] A. Canteaut and F. 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 Transactions on Information Theory*, vol. 44, no. 1, pp. 367–378, 1998.

[44] M. Finiasz and N. Sendrier, “Security bounds for the design of code-based cryptosystems,” in *International Conference on the Theory and Application of Cryptology and Information Security*, ser. Lecture Notes in Computer Science, M. Matsui, Ed., vol. 5912. Tokyo, Japan: Springer, Dec. 2009, pp. 88–105.

[45] D. J. Bernstein, T. Chou, and P. Schwabe, “McBits: Fast constant-time code-based cryptography,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, G. Bertoni and J. Coron, Eds., vol. 8086. Santa Barbara, CA, USA: Springer, Aug. 2013, pp. 250–272.

[46] T. Chou, “McBits revisited: toward a fast constant-time code-based KEM,” *Journal of Cryptographic Engineering*, vol. 8, no. 2, pp. 95–107, 2018.

[47] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, “pqm4: Testing and benchmarking NIST PQC on ARM cortex-m4,” *IACR Cryptology ePrint Archive*, p. 844, 2019.

[48] F. Standaert and C. Archambeau, “Using subspace-based template attacks to compare and combine power and electromagnetic information leakages,” in *International Workshop on Cryptographic Hardware and Embedded Systems*, ser. Lecture Notes in Computer Science, E. Oswald and P. Rohatgi, Eds., vol. 5154. Washington, D.C., USA: Springer, Aug. 2008, pp. 411–425.

[49] B. Hettwer, S. Gehrer, and T. Güneysu, “Applications of machine learning techniques in side-channel attacks: a survey,” *Journal of Cryptographic Engineering*, vol. 10, no. 2, pp. 135–162, 2020.

[50] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[51] C. O’Flynn and Z. Chen, “ChipWhisperer: An open-source platform for hardware embedded security research,” in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, ser. Lecture Notes in Computer Science, E. Prouff, Ed., vol. 8622. Paris, France: Springer, Apr. 2014, pp. 243–260.

[52] E. Prouff and M. Rivain, “Masking against side-channel attacks: A formal security proof,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, T. Johansson and P. Q. Nguyen, Eds., vol. 7881. Athens, Greece: Springer, May 2013, pp. 142–159.