Side Channel Information Set Decoding

Plaintext Recovery from the "Classic McEliece" Hardware Reference Implementation

Norman Lahr[1], Ruben Niederhagen[1], Richard Petri[1], and Simona Samardjiska[2]

[1] Fraunhofer SIT, Darmstadt, Germany
norman@lahr.email,ruben@polycephaly.org,rp@rpls.de
[2] Radboud Universiteit, Nijmegen, The Netherlands
simonas@cs.ru.nl

Keywords: ISD · Reaction Attack · SCA · FPGA · PQC · Niederreiter · Classic McEliece

Abstract. This paper presents an attack based on side-channel information and information set decoding on the Niederreiter cryptosystem and an evaluation of the practicality of the attack using a physical side channel. First, we describe a basic plaintext-recovery attack on the decryption algorithm of the Niederreiter cryptosystem. Our attack is an adaptation of the timing side-channel plaintext-recovery attack by Shoufan et al. from 2010 on the McEliece cryptosystem using the non-constant time Patterson's algorithm for decoding. We then enhance our attack by utilizing an Information Set Decoding approach to support the basic attack and we introduce column chunking to further significantly reduce the number of required side-channel measurements. Our practical evaluation of the attack targets the FPGA-implementation of the Niederreiter cryptosystem in the NIST submission "Classic McEliece" with a constant time decoding algorithm and is feasible for all proposed parameters sets of this submission. The attack idea is to distinguish between successful and failed error correction based on the Hamming weight of the decrypted plaintext using the electromagnetic field as side channel. We theoretically estimate that our attack improvements have a significant impact on reducing the number of required side-channel traces. We confirm our findings experimentally and run successful attacks against the "Classic McEliece" NIST submission parameter sets. E.g., for the 256bit-security parameter set `kem/mceliece6960119` we require starting from a basic attack with 6962 traces over a plain ISD approach with 5415 traces down to on average about 606 traces to mount a successful plaintext recovery attack.

1 Introduction

Many fields of research and industry are having high hopes on the power of quantum computing, e.g., for artificial intelligence, drug design, traffic control, and weather forecast [Mar17]. This growing interest in quantum computing has led

to a rapid development of quantum computers in the last decade. At the Consumer Electronics Show (CES) in 2019, IBM announced their first commercial quantum computer with 20 qubits [Nay19]. Even larger experimental quantum computers are operating in the labs of Google, IBM, and Microsoft. However, besides the high hopes in a new area of quantum computing, quantum computers pose a severe threat on today's IT security: A sufficiently large and stable quantum computer can give a quadratic speedup for brute-force attacks on symmetric cryptographic schemes like AES and SHA using Grover's quantum-computer algorithm [Gro96]. This threat is relatively moderate and can be mitigated by doubling security parameters, e.g., from AES-128 to AES-256. However, quantum computers can solve the integer factorization and discrete logarithm problems in polynomial time using Shor's quantum-computer algorithm [Sho94,Sho99], thus completely breaking most of the current asymmetric cryptography like RSA, DSA, and DH as well as ECC schemes like ECDSA, and ECDH.

As an answer to this threat on asymmetric cryptography, the research field of Post-Quantum Cryptography (PQC) has emerged in the last two decades, developing and strengthening alternative cryptographic schemes that are able to withstand attacks by quantum computers. The most popular approaches are hash-, lattice-, code-, and isogeny-based as well as multivariate cryptography [BBD09,FJP14]. Hash-based cryptography is used for very reliable signature schemes. An example is the IETF standard XMSS [BDH11,HBG+18]. Lattice-based cryptography has the reputation of being very efficient. However, code-based cryptography using certain codes is often regarded as already more mature and reliable. Conservative but less efficient examples for code-based cryptography are the McEliece [McE78] and the Niederreiter [Nie86] cryptosystems using binary Goppa codes. Multivariate schemes tend to be less popular due to efficiency issues. Isogeny-based cryptography is the most juvenile class of PQC and thus not yet fully trusted. In November 2017, the National Institute of Standards and Technology (NIST) started a public process for the standardization of PQC schemes; schemes from all classes mentioned above have been submitted.

An important question in the standardization process besides the definition of secure schemes and the choice of secure parameters is the impact of the implementation of a scheme on its security. A general requirement on the implementation of a scheme is that the runtime of the operations, e.g., key generation, signing, or decryption, does not vary based on secret information like the private key or the plaintext, i.e., a that the scheme has a constant-time implementation. (Constant time in regard to public input data like the public key or the ciphertext is not required for this property.) However, there are more side channels besides timing that might enable an attacker to get access to private information. Other side channels include power consumption and electromagnetic, photonic, and acoustic emission. For many PQC schemes, it is still unknown what side-channel attacks are practically feasible and how to protect against them. A general overview of the state of attacks on the implementation of PQC schemes is presented in [TE15]. In this work, we focus on one of the conservative (i.e., well understood and trusted) candidates in the NIST standardization process,

the "Classic McEliece" cryptosystem, and describe a plaintext-recovery attack on its decryption algorithm. The "Classic McEliece" cryptosystem — though honouring Robert McEliece, on of the pioneers of the code-based crypto field, with its name — is using the approach proposed by Niederreiter as described in Section 2.2.

Related Work. In [SSMS10] a timing attack on the McEliece PKC is presented that recovers the plaintext of a given ciphertext using a decryption oracle. In this attack, a bit-flip error is added to the ciphertext, which results in a shorter timing during decryption, if the flipped bit was set in the original error vector. Fault attacks on the variables used during encryption by McEliece and Niederreiter schemes are examined in [CD10]. A Differential Power Analysis (DPA) attack is presented in [CEvMS16] which recovers the secret key of a QC-MDPC McElice FPGA implementation by measuring the leakage of the carry occuring during the key rotation operation. A similar attack on a software implementation is presented in [FJP14], using the detection of counter overflows. An attack described in [RHHM17] uses rough information gained by DPA about the positions of set bits to recover the secret key in a cryptanalytic attack. A reaction attack is described in [SSPB19], where an attacker enhances the cryptanalysis of cryptosystems based on LRPC codes by exploiting their high decoding failure rate.

Information set decoding (ISD) is a well known decoding technique dating back to the work of Prange [Pra62] in the 1960's. In essence, in ISD, one tries to find a subvector (also referred to as Information set) of the error vector such that it has some predetermined pattern. For example, in the most basic variant by Prange - plain ISD, one tries to find a subvector that does not contain any errors, and contains enough information (hence the name information set) to decode uniquely a given word. Once an information set is found, decoding can be done using simple linear algebra.

This basic approach has been improved throughout the years: Lee and Brickell [LB88] proposed to allow errors in the information set which resulted in a polynomial improvement of the plain ISD. This was followed by minor improvements by Leon [Leo88]. Stern [Ste89] (and concurrently Dumer [Dum89]) first proposed to use *collision decoding* (actually the term was introduced later [BLP11]) where the search for the right information set is reduced to looking for collisions on sums of few columns of the parity check matrix. All subsequent improvements build on top of Stern's algorithm by exploring more refined techniques for collision search. The list is extensive and includes: [FS09,BLP11,MMT11,BJMM12,MO15].

We are not aware of a previous work that combines information set decoding with side channel analysis.

Our Contributions. In this work, we show how to adapt the side-channel attack from Shoufan et al. in [SSMS10] for plaintext recovery on the McEliece cryptosystem to a side-channel attack on the Niederreiter cryptosystem. We further improve our attack utilizing Information Set Decoding. Furthermore, we optimize the number of required side-channel queries by introducing an ap-

proach that works based on chunks. We demonstrate the feasibility of our attack using the hardware implementation by Wang et al. in [WSN18], which is accompanying the NIST submission "Classic McEliece" [BCL+17], exploiting an electromagnetic field (EM) side channel.

Structure. Section 2 provides a some background information on the code-based McEliece and Niederreiter cryptosystems and on Information Set Decoding (ISD). Section 3 follows up with a brief introduction to the side-channel attack from Shoufan et al. in [SSMS10], a description of our adaption of that attack to Niederreiter, and our improvements for reducing the number of queries. In Section 4, we provide a leakage analysis of the FPGA implementation from [WSN18] using simulated power traces, our measurement and analysis methods for the practical attack, as well as an evaluation of our approaches. Section 5 concludes the paper.

Notation. In the following $\mathrm{w}(x)$ denotes the function returning the Hamming weight (HW) of an input vector x. Capital letters like H denote matrices. Small letters denote integer values (e.g., m, n, k, i, j, and t) or vectors (e.g., plaintext p, ciphertext c, and error vector e). $\mathrm{GF}(p^q)$ denotes the Galois field for a prime p with extension degree q.

2 Background

In this section, we briefly introduce the McEliece cryptosystem as well as the Niederreiter cryptosystem including key generation, encryption, and decryption.

2.1 McEliece Cryptosystem

In 1978, McEliece proposed a cryptosystem using error correcting codes [McE78]. The basic idea of this cryptosystem is to use an efficient error correction algorithm that can correct up to t errors as secret key and an obfuscated generator matrix of the corresponding code as public key. With code length n and code dimension k, the public key is a $k \times n$ generator matrix G. Encryption works by computing a code word for the plaintext p using the generator matrix and by adding an error e with $\mathrm{w}(e) \leq t$ that is small enough so that the error correction algorithm is able to correct the error. The ciphertext c is therefore computed as $c = Gp + e$. The receiver simply corrects the error by applying his secret error correction algorithm, and recovers the plaintext from the code word.

2.2 Niederreiter Cryptosystem

In 1986, Niederreiter proposed a dual variant of the McEliece cryptosystem using a $(n-k) \times n$ parity-check matrix H instead of a generator matrix as public key [Nie86]. In this case, an error vector e of the weight $\mathrm{w}(e) = t$ is the plaintext; the syndrome $c = He$ of the error vector is the ciphertext. Here, an efficient

syndrome decoding algorithm is used for decryption. Due to the format requirements on the plaintext of having a certain length and weight, this scheme is usually used as a hybrid scheme with a random error vector that is used with a key derivation function to obtain a symmetric key for the encryption of the actual message.

In general, any error correcting code can be used for the McEliece and Niederreiter cryptosystems; however, in order to obtain an efficient and secure system, the code must be efficient to decode in possession of the secret key and hard to decode given only the public key and a ciphertext. McEliece proposed to use binary Goppa codes, which is still considered secure, while Niederreiter originally proposed to use Reed-Solomon codes, which turned out to be insecure [SS92]. Today, there are many variations of the McEliece and Niederreiter systems using different codes with different properties. However, using binary Goppa codes (for both McEliece and Niederreiter) is generally the most conservative choice. A drawback of using binary Goppa codes is the large size of the public key of around 1 MB for 256-bit security. In the following, we will focus on the Niederreiter cryptosystem with binary Goppa codes with parameters as defined in the NIST submission "Classic McEliece" for Round 1 [BCL+17] and Round 2 [BCL+19] (see also [BCS13]).

Key generation of the Niederreiter cryptosystem using binary Goppa codes works as follows (following the notation in [WSN17]): Choose a random irreducible polynomial $g(x)$ over $\mathrm{GF}(2^m)$ of degree t and a list $(\alpha_0, \alpha_1, \dots \alpha_{n-1}) \in \mathrm{GF}(2^m)^n$ of distinct elements of $\mathrm{GF}(2^m)$. From $g(x)$ and $(\alpha_0, \alpha_1, \dots \alpha_{n-1})$, compute the $t \times n$ parity-check matrix H as

$$H = \begin{bmatrix} 1/g(\alpha_0) & 1/g(\alpha_1) & \cdots & 1/g(\alpha_{n-1}) \\ \alpha_0/g(\alpha_0) & \alpha_1/g(\alpha_1) & \cdots & \alpha_{n-1}/g(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{t-1}/g(\alpha_0) & \alpha_1^{t-1}/g(\alpha_1) & \cdots & \alpha_{n-1}^{t-1}/g(\alpha_{n-1}) \end{bmatrix}$$

over $\mathrm{GF}(2^m)$. Transform H into a $mt \times n$ binary matrix H' by replacing each $\mathrm{GF}(2^m)$-entry by a m-bit column. Finally compute the systematic form $[\mathbb{I}_{mt}|K]$ of H' and return $g(x)$ and $(\alpha_0, \alpha_1, \dots \alpha_{n-1})$ as private key and K as public key. The last step of computing the systematic form of the binary parity-check matrix H' compresses the size of the public key from mtn bits to $mt(n - mt)$ bits, because the preceding identity matrix $\mathbb{I}_{mt}$ does not need to be stored or communicated.

Encryption works as follows: The sender constructs the $tm \times n$ binary parity-check matrix $H'' = [\mathbb{I}_{mt}|K]$ by appending K to the identity matrix $\mathbb{I}_{mt}$ and encrypts the error vector $e \in \mathrm{GF}(2)^n$ (i.e., the plaintext) with $\mathrm{w}(e) = t$ to the syndrome $s \in \mathrm{GF}(2)^{mt}$ as $s = H''e$ (i.e., the ciphertext).

Decryption of the syndrome depends on the error-correcting code that is used. Examples are Paterson's algorithm and the constant-time Berlekamp-Massey (BM) algorithm. Using the BM algorithm as in [BCS13,WSN18], decryption works as follows: Since the BM algorithm can only correct up to $t/2$ errors, use

Symbol	Description	Parameter Sets `kem/mceliece-` `6960119` `8192128`
$m \in \mathbb{N}$	Size of the binary field.	$m = 13$ $m = 13$
$t \in \mathbb{N}$	Correctable errors.	$t = 119$ $t = 128$
$n \in \mathbb{N}$	Code length.	$n = 6960$ $n = 8192$
$k \in \mathbb{N}$	Code dimension ($k = n - mt$).	$k = 5413$ $k = 6528$
$g(x)$	Goppa polynomial over $\mathrm{GF}(2^m)$ of degree t.	
$(\alpha_0, \alpha_1, \ldots \alpha_{n-1}) \in \mathrm{GF}(2^m)^n$	Support of n distinct elements of $\mathrm{GF}(2^m)$.	
$H \in \mathrm{GF}(2^m)^{t \times n}$	Parity-check matrix.	
$H', H'' \in \mathrm{GF}(2)^{mt \times n}$	Binary parity-check matrix.	
$H^{(2)} \in \mathrm{GF}(2^m)^{2t \times n}$	Double-size parity-check matrix.	
$K \in \mathrm{GF}(2)^{mt \times (n-mt)}$	Public key.	
$e \in \mathrm{GF}(2)^n$	Error vector (plaintext).	
$s \in \mathrm{GF}(2)^{mt}$	Syndrome (ciphertext).	
$s^{(2)} \in \mathrm{GF}(2^m)^{2t}$	Double-size syndrome.	
$\sigma(x)$	Error locator polynomial over $\mathrm{GF}(2^m)$ of degree t.	

Table 1. Symbols and parameter sets of Niederreiter for 256-bit security [BCL+19,WSN18].

the trick attributed to Sendrier in [HG13] and compute the double-size $2t \times n$ parity-check matrix

$$H^{(2)} = \begin{bmatrix} 1/g^2(\alpha_0) & 1/g^2(\alpha_1) & \cdots & 1/g^2(\alpha_{n-1}) \\ \alpha_0/g^2(\alpha_0) & \alpha_1/g^2(\alpha_1) & \cdots & \alpha_{n-1}/g^2(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{2t-1}/g^2(\alpha_0) & \alpha_1^{2t-1}/g^2(\alpha_1) & \cdots & \alpha_{n-1}^{2t-1}/g^2(\alpha_{n-1}) \end{bmatrix}$$

over $\mathrm{GF}(2^m)$. Then compute the double-size syndrome $s^{(2)} = H^{(2)} \cdot (s|0)$ by appending $n-mt$ zeros to the syndrome s. Now, we can use the BM algorithm to compute the error-locator polynomial $\sigma(x)$ of $s^{(2)}$. The roots of $\sigma(x)$ correspond to the error-positions. Therefore, the error-vector bits can be determined by evaluating $\sigma(x)$ at all points in $(\alpha_0, \alpha_1, \ldots \alpha_{n-1})$. If $\sigma(\alpha_i) = 0,\ 0 \leq i < n$, the ith bit of the error vector $e_i = 1$ or otherwise $e_i = 0$.

Table 1 shows the parameters `kem/mceliece6960119` and `kem/mceliece8192128` proposed by [BCL+19] for a security level of 256 bit (NIST category 5, i.e., 128-bit "post-quantum" security) and the symbols used in the above description following [WSN18].

2.3 Information Set Decoding

Suppose we are given a parity check matrix H and a syndrome s. An ISD algorithm solves the decoding problem:

$$\textit{Find } e,\ w(e) = w \textit{ such that } H \cdot e = s. \tag{1}$$

Basically, the algorithm tries to guess the error vector on k coordinates, and then uses this information to obtain the rest of the error coordinates. We call this set of k coordinates the *information set*, since it carries enough information to recover the entire error vector. Indeed, the decoding problem gives rise to the system:

$$\begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n} \\ h_{2,1} & h_{2.2} & \cdots & h_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k,1} & h_{n-k,2} & \cdots & h_{n-k,n} \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{n-k} \end{bmatrix} \tag{2}$$

with the error coordinates $e_1, \ldots, e_n$ as unknowns. If k coordinates are correctly guessed, the system (2) can be uniquely solved. We check the correctness of the solution by measuring the weight of the error. If the guess was wrong, we guess again.

Information set decoding was proposed by Prange [Pra62], and is also known as plain ISD. In this simplest form, we assume an error-free information set. The probability that we guess k error-free coordinates is $\binom{n-k}{w}/\binom{n}{w}$.

Stern's variant [Ste89] first introduced collision decoding that makes use of the birthday paradox. In essence, we allow some errors in the information set which increases the probability of success. The information set is split into sets with equal amount of errors p. Then the algorithm searches for collisions on these two sets, such that the sum of p columns restricted to ℓ coordinates matches the appropriate coordinates of the syndrome. It is the birthday decoding idea that improves asymptotically with respect to the previous variants. This idea was further generalized in the May-Meurer-Thomae (MMT) [MMT11] and the Becker-Joux-May-Meurer (BJMM) [BJMM12] variants that use the more elaborate generalized birthday problem. Here instead of looking for collisions between two lists, the collision search is between 4 or 8 lists in multiple layers. May and Ozerov [MO15] noticed that Stern's approach can be improved by using more sophisticated algorithms for approximate matching. Their approach is general enough to be applied to other variants such as BJMM.

3 Reaction-based Side-Channel Analysis

In this section, we describe our plaintext-recovery attack on the Niederreiter cryptosystem. In Section 3.1, we first briefly introduce the timing side-channel attack by Shoufan et al. from [SSMS10] on the McEliece cryptosystem. Then we explain how we adapt this attack to the Niederreiter cryptosystem in Section 3.2.

We describe how to reduce the number of queries required for a side-channel attack when using ISD in Section 3.3 and we improve our basic attack in Section 3.4 using larger chunks in each query. Finally, we explain how to combine the ISD techniques with our improved attack in Section 3.5.

3.1 Side-Channel Attack on McEliece

Shoufan et al. describe a plaintext recovery attack on the McEliece cryptosystem in [SSMS10] that is based on distinguishing the number of added error bits during the decoding step: The idea of the attack is to add (xor) an additional error bit to a given ciphertext at a certain position. If previously there had not been an error added to the code word on that position, in total, there is now one more error added to the code word. If previously there had already been an error at that position, the error is extinguished and there is now one error less. If the attacker is able to distinguish these two cases based on some side channel, he is able to mount the following attack: By iteratively adding an error to each position of the ciphertext and determining via the side channel if in total the number of errors has increased or decreased, the attacker is able to determine the position of all error bits, to correct the errors, and to decode the ciphertext.

Patterson's algorithm is a popular decoding algorithm for binary Goppa codes. However, the runtime of Patterson's algorithm depends on the number of errors that have been added to the code word. Shoufan et al. are using these timing variations in Patterson's algorithm as side-channel information to mount their attack: If an error is added to a previously error-free position, Patterson's algorithm has a slightly longer runtime; if an error is extinguished by the additional error bit, the runtime of Patterson's algorithm is slightly shorter. Precisely measuring and categorizing the runtime of Patterson's algorithm gives the required information to recover the error positions.

3.2 Side-Channel Attack on Niederreiter

In the attack by Shoufan et al. in [SSMS10] on the McEliece cryptosystem, the number of errors in the ciphertext is modified simply by adding one more error on varying positions to the original ciphertext. However, the Niederreiter cryptosystem is not operating with erroneous code words as ciphertext but with syndromes. The equivalent of adding an error to a code word in McEliece here is to add a column of the parity-check matrix (i.e., the public key) to the syndrome (i.e., the ciphertext). Therefore, we adapt the attack from [SSMS10] to the Niederreiter cryptosystem by systematically adding columns of the public key one by one to the original syndrome. If the bit corresponding to the column was not set in the original error vector (i.e., the plaintext), the number of errors in the modified syndrome is increased. Accordingly, if the corresponding bit was set, an error in the original error vector is effectively removed from the syndrome, reducing the number of errors. If an attacker can find a side channel that enables him to distinguish these two cases, he is able to mount an attack. Algorithm 1 shows the general approach for this attack. In order to distinguish the cases with

Algorithm 1: Iterative Reaction-based SCA

input : Classic McEliece parameters $n, m, t \in \mathbb{N}^+$,
binary parity-check matrix $H'' = [\mathbb{I}_{mt}|K] := (h_{i,j}) \in \mathrm{GF}(2)^{mt \times n}$,
syndrome $s \in \mathrm{GF}(2)^{mt}$.
output: Error vector $e \in \mathrm{GF}(2)^n$.

```
1 e ← (0, ..., 0);
2 for i ← 0 to n − 1 do
3 |   s' ← s ⊕ H''[i];
4 |   if QueryOracle(s') = true then
5 |   |   e[i] ← 1;
6 |   end
7 end
8 return e;
```

a reduced number of errors from the cases with an increased number of errors, a query to an oracle is required (line 4 in Algorithm 1) that returns *true* if the number is reduced and thus an error position has been found.

This decryption oracle can practically be achieved by having the victim decrypt the manipulated ciphertext and by measuring the side channel during the decryption. Therefore, when a non-constant time decoding algorithm like Peterson's algorithm is used, a timing side-channel attack as in [SSMS10] can be mounted on Niederreiter as well. In modern implementations, often a constant time algorithm, typically the BM decoding algorithm, is used in order to prevent timing side channels. Thus, in this case another side channel is required to mount the attack. In Section 4, we investigate EM side channels in the reference hardware implementation by Wang et al [WSN18] using the BM algorithm to demonstrate a practical attack. Another side-channel could for example be a response in a communication protocol if adding an error results in a decoding failure and if this failure is reported over the network.

The number of side-channel measurements that is needed for this basic iterative attack algorithm is the number of columns n in the parity check matrix, which is a few thousand for practical Niederreiter parameters (see Section 2.2). However, depending on the attack scenario, the attacker might only be able to take a limited amount of measurements, e.g., due to the cost of each measurement, limited access to the device, or additional countermeasures on the device. In the next sections, we describe improvements to this basic algorithm that allow the attacker to significantly reduce the number of decoding operations that he needs to query from the device under attack.

3.3 Reducing the Number of Queries with Information Set Decoding

The reaction attack described in Algorithm 1 recovers the entire error vector (all error coordinates) using side channel information. Thus, in order to be able

to decode, we need to collect at least n queries from the decryption oracle, which is 6960 and 8192 queries for the NIST parameters of "Classic McEliece" given in Table 1. However, we can reduce the number of queries substantially by focusing on the recovery of an information set of size $k < n$, instead. Once an information set is known, we can easily recover the entire error vector using basic linear algebra (see Section 2.3). The information set can be recovered from obtained queries or using some of the ISD algorithms described in Section 2.3. We can also combine the two techniques — first collect a number of queries, use them to reduce the problem to a smaller one, and then solve the smaller problem using an ISD algorithm.

In more detail, let $\mathrm{ISD}(n, k, w, H, s)$ be any ISD algorithm, such as Stern's or Ball Collision decoding, that on input of a parity check matrix $H \in \mathrm{GF}(2)^{(n-k)\times n}$ and syndrome $s \in \mathrm{GF}(2)^{n-k}$ outputs an error vector $e \in \mathrm{GF}(2)^n$ — a solution to the decoding problem (1).

Suppose we are given an oracle as in Algorithm 1 that we can use to learn the value of a coordinate e_i of the error vector. Using the oracle, we learn a subset of error indices $\widehat{E} \subset E = \{1, \ldots, n\}$ We denote the corresponding subvector of e by $\hat{e} = (e_i)_{i\in\widehat{E}}$ and its complement by $\tilde{e} = (e_i)_{i\in E\setminus\widehat{E}}$. We split similarly the columns $h_i = (h_{j,i})_{j\in\{1,\ldots,mt\}}$ of the matrix H''. Let $\widehat{H}'' = (h_i)_{i\in\widehat{E}}$ and $\tilde{H}'' = (h_i)_{i\in E\setminus\widehat{E}}$. With this notation, from the obtained information, we can transform the decoding problem (1) to:

$$\tilde{H}'' \cdot \tilde{e} = \tilde{s} \tag{3}$$

where $\tilde{s} = s - \widehat{H}'' \cdot \hat{e}$.

So we have reduced our initial problem to a smaller decoding problem with parameters $k' = k - |\widehat{E}|$, $n' = n - |\widehat{E}|$, $w' = w - \mathrm{w}(\hat{e})$. We solve this problem by calling the available ISD algorithm $\mathrm{ISD}(n', k', w', \tilde{H}'', \tilde{s})$. Note that, if $|\widehat{E}| = k$, we have recovered an information set, and we only need to solve a linear system using Gaussian elimination. Thus, for convention, we assume $\mathrm{ISD}(n, 0, w, H, s)$ simply calls Gaussian elimination procedure. Algorithm 2 details the whole procedure.

The performance of Algorithm 2 depends directly on the size of set $\widehat{E}$, i.e., on the number of queries to the oracle. There is a clear trade-off between the running time and the queries to the oracle, which is depicted in Figure 1. Basically, the attacker is free to choose the number of queries that they perform based on their computational resources.

In our depiction of the trade-off, for simplicity, we used only two ISD algorithms — Stern's and MMT. We did not use the state of the art BJMM variant, because there is no compact representation of the concrete complexity of this algorithm.

3.4 Reducing the Number of Queries with Iterative Chunking

For the approach that we describe here, we need to slightly change the oracle from the previous Section. In particular we assume the oracle returns *true* if

Algorithm 2: ISD-supported Iterative Reaction-based SCA

input : Classic McEliece parameters $n, m, t \in \mathbb{N}^+$,
binary parity-check matrix $H'' = [\mathbb{I}_{mt}|K] := (h_{i,j}) \in \mathrm{GF}(2)^{mt \times n}$,
syndrome $s \in \mathrm{GF}(2)^{mt}$.
output: Error vector $e \in \mathrm{GF}(2)^n$.

$\widehat{E} \subset E = \{1, \ldots, n\}, |\widehat{E}| \leqslant k$;
$e \leftarrow (0, \ldots, 0)$;
for $i \in \widehat{E}$ **do**
  $s' \leftarrow s \oplus H''[i]$;
  if QueryOracle$(s') = true$ **then**
    $e[i] \leftarrow 1$;
  end
end
$\tilde{s} \leftarrow s - \widehat{H}'' \cdot \hat{e}$;
$\tilde{e} := (e_i)_{i \in E \setminus \widehat{E}}$;
$\tilde{H}'' := (h_i)_{i \in E \setminus \widehat{E}}$;
$\hat{e} := (e_i)_{i \in \widehat{E}}$;
$\tilde{e} \leftarrow \mathrm{ISD}(n - |\widehat{E}|, k - |\widehat{E}|, w - \mathrm{w}(\hat{e}), \tilde{H}'', \tilde{s})$;
$e \leftarrow \mathrm{Reconstruct}(\hat{e}, \tilde{e})$;
return *e;*

the number of errors has not increased (instead of reduced as in Section 3.2) and false otherwise. Note that the real oracle that we construct in Section 4.2 actually captures both cases.

To get some intuition on how our *iterative chunking* works, we first present a simpler variant that already reduces the number of needed queries by more than 35%.

Suppose that instead of a single error index, we query two error indices (a chunk of two) at once. We first randomly select the chunk (i, j) of error indices, $i, j \in \{1, \ldots, n\}, i \neq j$. We add both columns h_i and h_j to the syndrome s, to obtain the new syndrome s'. We give the input s' to the decryption oracle. Notice that the decryption oracle will output *false* only in the case when the values at corresponding error indices in the error vector were $(e_i, e_j) = (0, 0)$ (a 'low' chunk). In all the other cases (we refer to them as 'high' chunks, to indicate that there is at least one '1' in the chunk) the decryption oracle will output *true*. Indeed, if $(e_i, e_j) = (0, 0)$, after adding the pair of columns (h_i, h_j) to the syndrome, we obtain $(e'_i, e'_j) = (1, 1)$, and in total $w + 2$ errors. Hence, the number of errors has increased and the decryption oracle will output *false*. If $(e_i, e_j) = (0, 1)$ or $(e_i, e_j) = (1, 0)$, we get $(e'_i, e'_j) = (1, 0)$ and $(e'_i, e'_j) = (0, 1)$ respectively, and in this case the number of errors does not change (it remains w) so the decryption oracle returns *true*. In the last case, $(e_i, e_j) = (1, 1)$, after adding the columns we obtain $(e'_i, e'_j) = (0, 0)$. So in this case, the number of

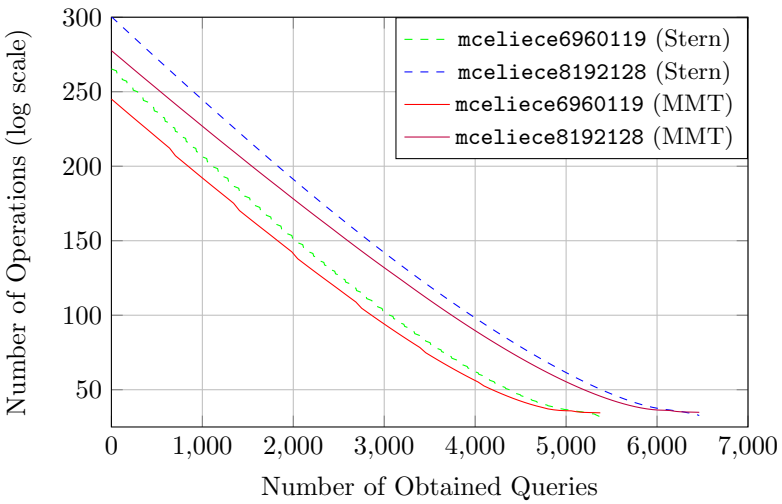

Fig. 1. Time-queries trade-off when using ISD decoding algorithms.

errors reduces to $w-2$, and the decryption oracle returns *true* as well. Table 2 summarizes the above.

What we can conclude from the previous is that if *false* is returned, we can be sure that the corresponding error positions in the error vector were $(e_i, e_j) = (0,0)$. Since the length of the error vector is much bigger than its Hamming weight, most of the time the randomly chosen chunk will be $(e_i, e_j) = (0,0)$, and we can confirm these values by only one query, instead of two as in the approach from the previous section. We perform the procedure for new random pairs of positions (i,j) until we find $k/2$ pairs whose initial state was $(0,0)$ i.e., until we encounter $k/2$ *false* oracle answers. Note that after a pair has been queried, we need to undo the changes made, i.e., return the pair to its initial state.

Proposition 1 can be used to estimate how many queries we need to perform (how many queries we need to collect) in order to learn $k = n - mt$ error positions, and to be able to uniquely solve system (3). For example, for the NIST parameters of Classic McEliece we need 3480 queries for `kem/mceliece6960119` and 4096 queries for `kem/mceliece8192128`.

We extend our idea to chunks $(e_{i_1}, \ldots, e_{i_\beta})$ of size β. We keep the convention of calling the all-zero chunk $(0, \ldots, 0)$ 'low' chunk and all other chunks containing 1s 'high' chunks. First note that we can not directly use the same approach for chunks of size $\beta \geqslant 2$. For example for $\beta = 3$ we have Table 3 analogous to Table 2.

Table 3 shows (columns 3 and 4) that there is ambiguity in the oracle answers, so we can not distinguish whether the chunk was $(0,0,0)$ or $(0,0,1)$; $(0,1,0)$; $(1,0,0)$. However, we can remedy this situation if we reduce the initial number of errors from w to $w-1$ as columns 5 and 6 from Table 3 show. This

initial	after	new $N_0(err)$	Oracle
$(0,0)$	$(1,1)$	$w+2$	false
$(0,1)$	$(1,0)$	w	true
$(1,0)$	$(0,1)$	w	true
$(1,1)$	$(0,0)$	$w-2$	true

Table 2. Overview of the output of the decryption oracle when querying chunks of two at once. The first column shows the initial state of the queried chunk, the second column shows the state of the pair after 'flipping' the values, the third column shows the total number of errors in the new state, and the last column shows the oracle answer.

initial	after	new $N_0(err)$ (w)	Oracle	new $N_0(err)$ $(w-1)$	Oracle
$(0,0,0)$	$(1,1,1)$	$w+3$	false	$w+2$	false
$(0,0,1)$	$(1,1,0)$	$w+1$	false	w	true
$(0,1,0)$	$(1,0,1)$	$w+1$	false	w	true
$(1,0,0)$	$(0,1,1)$	$w+1$	false	w	true
$(1,1,0)$	$(0,0,1)$	$w-1$	true	$w-2$	true
$(1,0,1)$	$(0,1,0)$	$w-1$	true	$w-2$	true
$(0,1,1)$	$(1,0,0)$	$w-1$	true	$w-2$	true
$(1,1,1)$	$(0,0,0)$	$w-3$	true	$w-4$	true

Table 3. Overview of the oracle answers for $\beta = 3$ when the initial number of errors is w and $w-1$.

requires knowledge about the position of one 1 in the error vector. Adding the corresponding column of the matrix H'' to the syndrome reduces the number of errors to $w-1$. We determine the position of one 1 by querying chunks of size $\beta = 2$ until a high chunk is found. Querying all positions within the high chunk we are certain reveals the position of the 1 (we will explain a more efficient divide and conquer strategy later on in Algorithm 4). The same reasoning extends to any chunk size β: If the number of errors before we start querying β size chunks is $w-(\beta-2)$, the oracle answers *false* only for low chunks, and we can use this information to distinguish low chunks.

Informally, we start with chunk size $\beta_1 = 2$ and continue iteratively. For a chunk size β_j, suppose we have n_{j-1} columns (positions in the error vector) left, and the number of errors remaining is w_{j-1}. We query random chunks without replacement until decoding succeeds. Success of the decoding indicates a high chunk. We inspect the positions within the high chunk, and locate all 1s. If at position i there is a 1, we add the corresponding column h_i of the matrix H'' to the syndrome. Suppose we found a total of L_j low chunks and the number of 1s found in the high chunk is ν_j. After this procedure, we have found $\beta_j L_j + \beta_j$ linear equations, the pool of available columns has reduced to $n_j = n_{j-1} - \beta_j L_j - \beta_j$, and the number of errors within those columns is $w_j = w_{j-1} - \nu_j$.

Next, we increase the chunk size to $\beta_{j+1} = \beta_j + \nu_j$, and repeat the same procedure. We continue until the chunk size reaches a threshold β_T. The threshold β_T is an optimization parameter, and we determine its value so that the number of necessary queries to recover an information set is minimized.

After the threshold is reached, we do not increase the chunk size any more. Now, note the following important observation: Since we don't need to increase the chunk size, it is not essential to find error positions; we only want to find enough low chunks so that we recover an information set. So in principle, we do not care about high chunks, unless there are too few chunks remaining – not enough to recover an information set. Therefore, instead of throwing away the high chunks, we save them in a bucket, and potentially inspect some of them if the number of columns in the bucket has surpassed $n - k$. This algorithm is given in pseudo-code in Algorithm 3. The algorithm for inspecting the high chunks and locating the set bits within the chunk is given in Algorithm 4. It uses a divide and conquer strategy to reduce the number of needed queries.

Later in this section, we argue that actually for the optimal T we don't need to use the columns in the bucket. Intuitively, in this case, we do not inspect the high chunks, which should reduce the total amount of necessary queries.

We next analyze the approach in order to determine the best threshold value, and estimate the number of queries needed for the attack. First let's make some important observations. In the first part of the attack, we iteratively increase the chunksize at each step, right after we find a high chunk. Let $E(\beta, n_\beta, w_\beta)$ denote the expected number of queries needed to find a high chunk of size β for n_β available columns of total weight w_β. Then, the number of error positions that we learn at this step is $\beta E(\beta, n_\beta, w_\beta)$. Repeating the same for chunksizes $\beta_1, \beta_2, \ldots, \beta_{T-1}$, we end up with $n_T = n - \sum_{i=2}^{\beta_{T-1}} iE(i, n_i, w_i)$ unqueried error positions, and we need to learn $k_T = k - \sum_{i=2}^{\beta_{T-1}} iE(i, n_i, w_i)$ more error positions in order to be able to solve uniquely the linear system (3). Furthermore, let $E(w(\beta))$ denote the expected weight of the found high chunk. Then, there is an expected number of $w_T = w - \sum_{i=2}^{\beta_{T-1}} E(w(i))$ errors among the unqueried n_T positions.

When we reach the threshold value β_T we change the strategy, and continue to query only chunks of size β_T. Suppose we make c_T queries. Clearly, $n_T \geqslant \beta_T c_T$. In order to be able to learn enough error positions in this part without inspecting any high chunks, we need $k_T \geqslant \beta_T E(c_T)$ where $E(c_T)$ is the expected number of low chunks among the c_T queried chunks. Of course, we could inspect high chunks as in Algorithm 3 if we do not obtain enough low chunks. However this is not the optimal strategy, since we need on average $2 \log \beta_T$ additional queries to learn β_T error positions, and this quickly becomes too expensive. Thus we need to find the maximal β_T such that $k_T \geqslant \beta_T E(c_T)$. We use the following simple result. The proof is given in Appendix A.

Proposition 1. *Let n be the length of the error vector, w be the weight of the error vector, and let β be the chunk size that we query. We query chunks without replacement, until we find a high chunk. The expected number of queries until*

Algorithm 3: Chunk-based Side-channel Attack

input : Classic McEliece parameters $n, m, t \in \mathbb{N}^+$,
binary parity-check matrix $H'' = [\mathbb{I}_{mt}|K] := (h_{i,j}) \in \mathrm{GF}(2)^{mt \times n}$,
syndrome $s \in \mathrm{GF}(2)^{mt}$,
threshold β_T.

output: Partial error vector $e' \in \mathrm{GF}(2)^n$,
bucket of chunks containing an error position,
list of remaining column indices.

$e' \leftarrow [0, \ldots, 0]$; // initialize with zero vector
$\beta \leftarrow 2$; // start with chunk size 2
$s'[1] \leftarrow s$; $s'[\beta] \leftarrow s$; // list of syndromes $s'[i]$ for each chunk size $0 < i \leq \beta_T$
indices $\leftarrow [n, \ldots, 0]$; // column indices in reverse order
bucket $\leftarrow []$;
while Len(*indices*) $> \beta$ **do**
  chunk $\leftarrow$ Pop(*indices*, β) ; // pop β-many indices
  $s'' \leftarrow s'[\beta]$ + Sum([$H''[i]$ ***for*** i ***in*** *chunk*]) ; // add columns in the chunk to s'
  if QueryOracle(s'') = *true* **then** // there is an error position
    if $\beta < \beta_T$ **or** $|bucket| > n - k$ **then** // find errors until threshold is reached
      $e_{id} \leftarrow$ FindErrorPositions(*chunk*, s', H'') ; // find errors in chunk
      for i **in** e_{id} **do**
        $e'[i] \leftarrow 1$; // update e' with found errors
        if $\beta < \beta_T$ **then**
          $s'[\beta + 1] \leftarrow s'[\beta] - H''[i]$; // remove found errors from syndrome
          $\beta \leftarrow \beta + 1$; // increase chunk size, up to β_T
        end
      end
    else
      bucket $\leftarrow$ *bucket* + *chunk* ; // collect chunks with remaining errors
    end
  end
end
return e', *bucket*, *indices*;

we find a high chunk of size β is:

$$E(\beta, n_\beta, w_\beta) = \frac{1}{\binom{n}{w}} \sum_{i \geqslant 0} \binom{n - i\beta}{w}. \tag{4}$$

The expected weight of the high chunk of size β is $E(w(\beta)) = \dfrac{w' \binom{n'-1}{\beta-1}}{\binom{n'}{\beta} - \binom{n'-w'}{\beta}}$.
where n' is the number of unqueried error positions before the occurrence of the high chunk, and w' is the total number of errors at these positions.

Algorithm 4: FindErrorPositions

input : $chunk \in \mathbb{N}^i$,
list of temporary syndromes s' with $s[i] \in \mathrm{GF}(2)^{mt}$,
binary parity-check matrix $H'' = [\mathbb{I}_{mt}|K] := (h_{i,j}) \in \mathrm{GF}(2)^{mt \times n}$.
output: List with error indices.

```
stack ← [FirstHalf(chunk), SecondHalf(chunk)];
e_id ← [];
while stack not empty do
    chunk ← Pop(stack);
    s'' ← s'[Len(chunk)] + Sum([H''[i] for i in chunk]);
    if Query(s'') = true then                    // there is an error position
        if Len(chunk) = 1 then
            Push(e_id, chunk[0]);
        else
            Push(stack, FirstHalf(chunk));
            Push(stack, SecondHalf(chunk));
        end
    end
end
return e_id;
```

Using the proposition we estimate the total number of queries for a given threshold β_T. Note that we use an algorithmic procedure since there is no compact accurate expression. For the two parameter sets `kem/mceliece6960119` and `kem/mceliece8192128` of the "Classic McEliece" submission (see Table 1) we estimated the expected number of queries for $\beta_T \in \{2, \ldots, 34\}$. The results are depicted in Figure 2. The optimal threshold value for `kem/mceliece6960119` is $\beta_T = 16$ (or $\beta_T = 17$) for which the amount of needed queries is around 610 and for `kem/mceliece8192128` the optimal threshold is again $\beta_T = 16$ (or $\beta_T = 17$) for which the amount of needed queries is around 700.

3.5 Combining Iterative Chunking with Information Set Decoding

The number of needed queries can be further decreased by combining Iterative chunking with some ISD algorithm. Instead of recovering an entire information set from queries, we can stop early, when we have learned only $\delta_k < k$ error coordinates. Assume at this point we have n' columns remaining, the weight of the error vector on these coordinates is w' and we need to recover $k' = k - \delta$ more elements from the information set.

Then we are left with the decoding problem with parameters (n', k', w') which we can solve using any ISD algorithm. Of course this comes with a price since ISD algorithms are exponential in time. Finding the right trade-off depends on the computational power (CPU hours) the attacker has at hand. Figure 3 gives the trade-off when using Stern's or MMT algorithm.

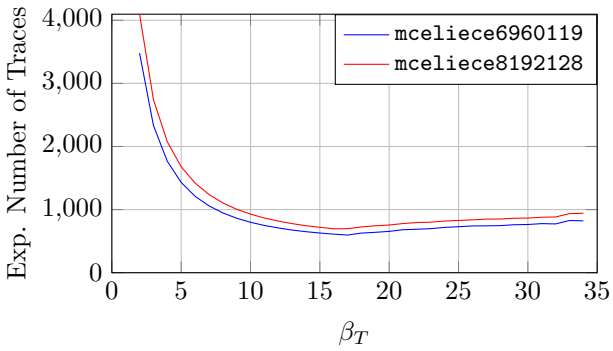

Fig. 2. The expected number of queries needed for threshold values from 2 to 35.

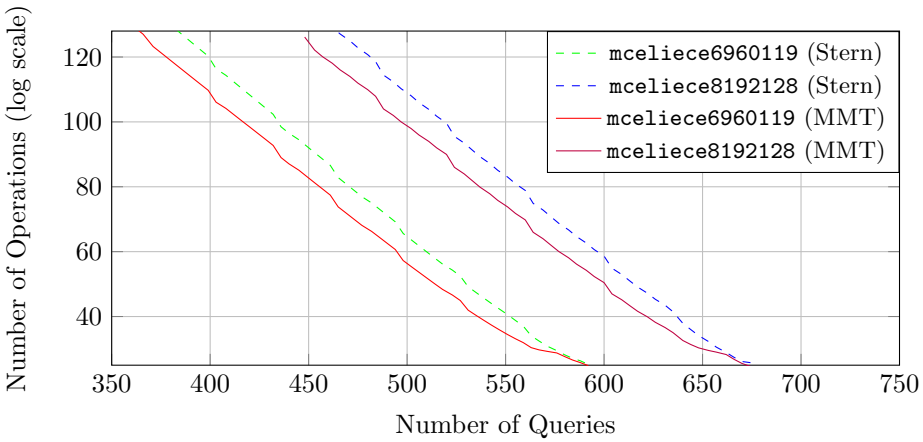

Fig. 3. Time-queries trade-off when using ISD decoding algorithms.

4 Attack Evaluation

For the practical attack evaluation we adapted the implementation presented in [WSN18] for Field Programmable Gate Arrays (FPGAs). In Section 4.1 we describe our approach for preliminary leakage analysis of the description module design and in Section 4.2 the construction of a practical decryption oracle. Finally, in Section 4.3 we evaluate the practical attack.

4.1 Leakage Analysis

To construct a decryption oracle for our attack approach we investigated the implementation by Wang et al. from [WSN18] in detail to find a proper point of interest at which we can find significant leakage. The selected implementation

uses the constant-time BM algorithm for the error correction. The BM algorithm returns an error-locator polynomial which has roots at the points that correspond to an error position. Thus, if there are $t' \leq t$ errors, t' input points to the error-locator polynomial evaluate to zero. If the number of errors is larger than t, a random polynomial is returned by the BM algorithm, which most likely has a very small number of roots. Thus, in order to distinguish whether the number of errors has increased or decreased, we need to distinguish cases where the reconstructed error vector has a low HW (for cases with an increased error number where error correction failed) and where it has a high HW (for cases with a decreased error number where error correction succeeded).

The FPGA implementation of the decryption module in [WSN18] consists of five major steps: First, an additive Fast Fourier Transformation (FFT) evaluates the secret Goppa polynomial. Then the double syndrome is computed. Afterwards the BM algorithm is performed and another additive FFT is applied to evaluate the error-locator polynomial. In the final step, the error vector is constructed.

In addition to the analysis of the source code we simulated the implementation for a preliminary leakage analysis in order to find possible leakage in a noise-free simulated environment. We wrote a Python script that computes a simulated power trace from a Value Change Dump (VCD) file of an Icarus Verilog (iverilog) simulation using a simple Hamming-distance model. This results in a simulated power trace with cycle accuracy. Figure 4 shows the resulting graphs of the simulation for two different simulated power traces, one for a successful decoding (high HW error vector) and one for an unsuccessful decoding (low HW error vector). The five steps of the decryption are highlighted. The first point at which side-channel information is leaked is at the last round of the second additive FFT operation which evaluates the error-locator polynomial from the BM decoder. The result of this evaluation equals to zero if there is a root and results in other values if not. Thus, it is distinguishable in general. However, because the implementation of the additive FFT utilizes several multiplicator instances in parallel, the logical noise added to the exploitable leakage is quite high.

The second point is at the last step, the error vector construction. Here, the graph of the high HW error vector (blue) increases during the construction in contrast to the low HW error vector (red) such that there is a growing distance between them. The reason for the increasing number of bit flips at this stage is that the result of the error vector construction is shifted into a large flip-flop shift-register step by step. To lower the effort compared to the analysis of the second FFT we exploit the significant leakage of the iterative reconstruction at the end of the decryption process. Therewith, the side-channel information enables the construction of a decryption oracle.

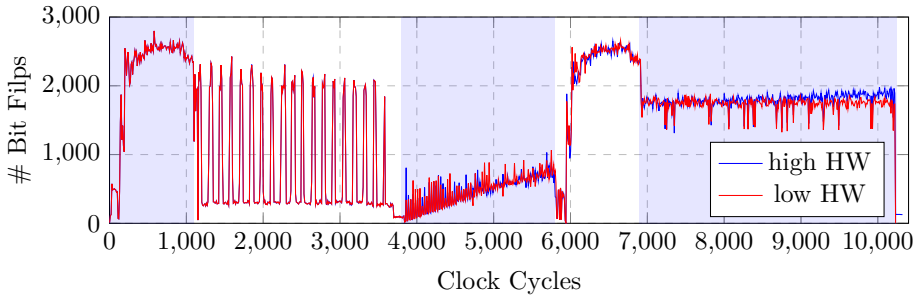

Fig. 4. Simulated power consumption based on a Hamming-distance model for successful decoding (high HW in the resulting error vector) and unsuccessful decoding (low HW) using a small parameter set with $m = 12$, $t = 66$, and $n = 3307$. The different parts of the algorithm can clearly be seen (marked with alternating blue background color). In the first block, an additive FFT is performed for evaluating the secret Goppa polynomial. In the second block, the double syndrome is computed. In the third block, BM is performed. In the fourth block, another additive FFT is performed in order to evaluate the error-locator polynomial. Finally, the error vector is constructed. A clear difference in the simulation is visible in the last step for the low and high HW results.

4.2 Building an Oracle in Practice

We decided to use the electromagnetic radiation (EMR) leakage emanated by the FPGA during the decryption and developed a Differential Electromagnetic Analysis (DEMA). Power leakage could be exploited in the same way.

To get a response for individual queried syndromes we apply Welch's t-test [Wel47] to compare the means of the traces from the error-vector construction range against two known reference traces. The reference traces stem from deciphering the original syndrome for which we know that it is decodable, and from a faulty syndrome that includes more than t errors so that it cannot be decoded. This has the disadvantage that it adds an overhead of two traces to the total number of required traces. Alternatively, we could statistically determine a threshold to compare it to the result of a t-test with just one of the reference traces, which would save one trace. However, we decided to spend one additional trace and avoid the statistical computation. The faulty syndrome is constructed by adding five columns randomly chosen from the public key matrix H''. The probability that this results in syndrome with more errors than can be corrected is high; nevertheless, we use a t-test comparison to the trace of the original syndrome to ensure this requirement. Algorithm 5 details the preparation procedure. Now, we take the difference of the p-values of the t-tests comparing a trace T_i against the original syndrome trace T_{high} and against the faulty syndrome trace T_{low}:

$$p_\Delta = \text{t-test}(T_i, T_{high}) - \text{t-test}(T_i, T_{low}) \tag{5}$$

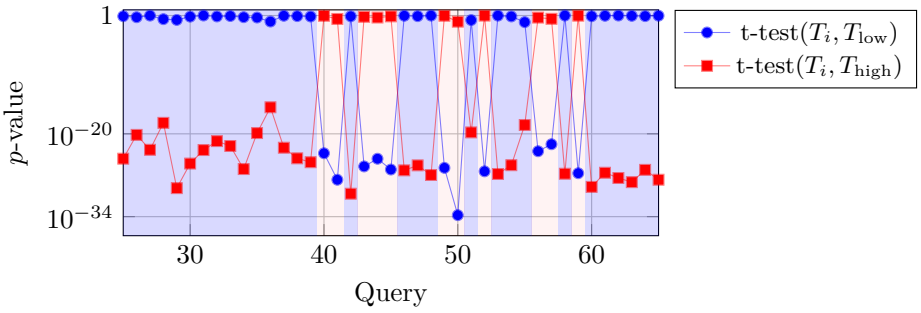

Fig. 5. Example p-values of the t-tests of consecutive oracle queries. For each query a trace T_i is compared to a known faulty (T_{low}) and known good trace (T_{high}). If a trace is similar to the faulty trace (i.e. due to higher p-value), a decoding failure is detected (light blue background). In the opposite case (light red background) the syndrome could be decoded.

Thus, if the p-value is positive the acquired trace is similar to the original syndrome trace and interpreted as decodable. Otherwise, if the p-value becomes negative it is interpreted as not decodable. Figure 5 gives an example section of p-values of consecutive oracle queries. The response when used as decryption oracle therefore is:

$$response = \begin{cases} true \text{ (decodable)}, & \text{if } p_\Delta > 0 \\ false \text{ (not decodable)}, & \text{otherwise} \end{cases} \tag{6}$$

Algorithm 6 shows a query process. This approach requires just a single trace per query plus two traces for the reference traces at the beginning. In detail, we cannot apply the t-test to the raw traces directly because of misalignment between the signals. To handle this, we apply trace compression similar as described in [MOP07]. We use the clock signal and take the maximum peak-to-peak difference of the amplitudes of the power or EM signal in each first clock half-wave as the new value. Instead of a t-test we could also apply an F-test to compare the variances of the raw traces, but the analysis duration is higher on our system in contrast to the t-test over the compressed traces.

4.3 Practical Evaluation

To evaluate the attack in practice, we ported the FPGA design of [WSN18] to a Xilinx Kintex-7 (XC7K160T) on a SAKURA-X[3] board running at 24 MHz clock frequency. We added a UART communication interface, a control unit that handles the storage of the secret key parts $(g(x), (\alpha_0, \alpha_1, \ldots \alpha_{n-1}))$, and a trigger signal to one of the output ports that indicates the start and the end

[3] `http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html`

Algorithm 5: GetReferences

input : Classic McEliece parameters $n, m, t \in \mathbb{N}^+$,
binary parity-check matrix $H'' = [\mathbb{I}_{mt}|K] := (h_{i,j}) \in \mathrm{GF}(2)^{mt \times n}$,
syndrome $s \in \mathrm{GF}(2)^{mt}$.
output: Reference traces T'_{high}, T'_{low}.

Decrypt$(s) \rightsquigarrow T_{high}, Clk_{high}$;
$T'_{high} \leftarrow$ Compress(T_{high}, Clk_{high});
$s^* \leftarrow s$;
repeat
  for $i \leftarrow 0$ **to** 5 **do**
    $i \in^R \{0 \dots n\}$;
    $s^* \leftarrow s^* \oplus H[i]$;
  end
  Decrypt$(s^*) \rightsquigarrow T_{low}, Clk_{low}$;
  $T'_{low} \leftarrow$ Compress(T_{low}, Clk_{low});
until t-test$(T'_{high}, T'_{low}) \approx 0$;
return T'_{high}, T'_{low};

Algorithm 6: DEMA-based QueryOracle

input : Manipulated syndrome $s' \in \mathrm{GF}(2)^{mt}$,
Reference traces T'_{high}, T'_{low}.
output: Oracle response.

Decrypt$(s') \rightsquigarrow T_j, Clk_j$;
$T'_j \leftarrow$ Compress(T_j, Clk_j);
$p_\Delta \leftarrow$ t-test$(T'_j, T'_{high}) -$ t-test(T'_j, T'_{low});
return $\begin{cases} true & \text{if } p_\Delta > 0 \\ false & \text{otherwise} \end{cases}$;

of a decryption operation. We acquired the EMR profiles and the clock signal using a LeCroy WavePro 715Zi oscilloscope at 500 MSamples/s and a near-field probe from Langer (RF-U 5-2). We added a 10 MHz high-pass filter to remove noise in the lower frequency range and used a customized Python script for an automatic acquisition and analysis process. The ISD-support was implemented using the GF(2) arithmetic of SageMath[4].

We analyzed the design of the Niederreiter decryption for the two parameter sets $(n, m, t) = \{(6960, 13, 119), (8192, 13, 128)\}$ as proposed for "Classic McEliece" in the second round of the NIST PQC competition (corresponding to `kem/mceliece6960119` and `kem/mceliece8192128`, see Table 1); our approach can be applied to the other "Classic McElice" parameters as well. Figure 6 shows

[4] `http://www.sagemath.org/`

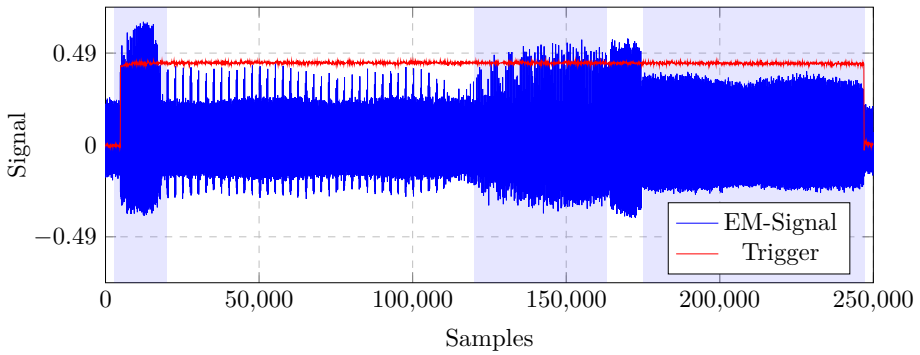

Fig. 6. Example EM trace of the decryption process on a SAKURA-X (Xilinx Kintex-7, XC7K160T). The five steps (additive FFT, double syndrome computation, Berlekamp-Massey, additive FFT, error vector construction) are highlighted with alternating blue background color as in Figure 4.

an example trace of a decryption run. Corresponding to the simulated trace in Figure 4, the five individual parts of the decryption process are identifiable.

We evaluated the ISD-supported iterative approach (Section 3.3) and the chunk-based approach (Section 3.4) using the oracle described in Algorithm 6 for both parameter sets. We are using plain ISD (Prange) as ISD algorithm, replacing the "guessing" with queries to our oracle, effectively implementing the ISD as Gaussian elimination on mt columns. If an attacker is able to invest some computing power, he is able to reduce the number of traces even further, trading in a higher complexity of the ISD. For each parameter set we tested five different key pairs using five different plain-/ciphertext pairs for each key pair (25 in total). We used the SageMath scripts which are enclosed with the publicly available FPGA implementation of the Niederreiter cryptosystem from Wang et al. to generate the required plain-/ciphertexts and key pairs using different seeds. According to the theoretical analysis result showed in Figure 2 we evaluated the chunk-based approach for both proposed optimal values for the chunksize threshold β_T and some additional values.

For each approach we were able to recover the entire plaintexts in all cases. Table 4 shows the results of the evaluation. For the ISD-supported iterative SCA approach we performed the required $k = n - mt$ measurements. The results of the chunk-based approach meet the predicted theoretical values. The results show that the number of traces can be lowered to approximately 10% compared to the ISD-supported iterative approach. As predicted, the number of traces depends on the choice of the chunksize threshold β_T. A smaller chunksize causes more queries. If the chunksize is too large the bucket gets filled up with not inspected, as high responded queries and additional traces have to be acquired to collect enough columns to apply plain ISD. We could also spend more computing power to apply Stern or MMT as mentioned above to lower the number of traces further.

kem/mceliece-	Approach	Statistics min.	avg.	max.	Theory
6960119	ISD-sup. Iterative	–	5415 $(k+2)$	–	
	Chunk-based, $\beta_T = 19$	576	637.0	702	640
	Chunk-based, $\beta_T = 18$	547*	610.0*	666*	628
	Chunk-based, $\boldsymbol{\beta_T = 17}$	**537**	**599.88**	**649**	**607**
	Chunk-based, $\beta_T = 16$	553	605.56	663	611
	Chunk-based, $\beta_T = 15$	571*	620.72*	682*	629
8192128	ISD-sup. Iterative	–	6530 $(k+2)$	–	
	Chunk-based, $\beta_T = 19$	652	723.72	803	744
	Chunk-based, $\beta_T = 18$	643	706.56	810	725
	Chunk-based, $\boldsymbol{\beta_T = 16}$	**637**	**694.84**	**770**	**696**
	Chunk-based, $\beta_T = 15$	660*	714.76*	790*	719

Table 4. Statistical data for the required number of traces for a successful recovery of an entire error vector gathered from 5 different key pairs and 5 different plain-/ciphertexts per key pair for each parameter set. The values marked with an asterisk are from a simulated oracle; we will provide values from the real oracle in the camera-ready version of this paper.

5 Conclusion

In this paper we have shown that plaintext recovery attacks using side-channel information against the Niederreiter cryptosystem and specifically the "Classic McEliece" NIST submission are feasible with a relatively small amount of queries and that therefore countermeasures are required if side channels can be exploited by an attacker. Using plain ISD and our improved attack strategy, we require less than 700 traces when attacking the reference FPGA implementation of the "Classic McEliece" submission even for the largest parameter sets. The number of traces can be further reduced by using state-of-the-art ISD algorithms in a trade-off with the computing time of ISD.

References

BBD09. Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors. *Post-Quantum Cryptography*. Springer, 2009.

BCL+17. Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wen Wang. Classic McEliece: conservative code-based cryptography — Round 1, November 2017. `https://classic.mceliece.org/nist/mceliece-20171129.pdf`.

BCL+19. Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, and Wen Wang. Classic

McEliece: conservative code-based cryptography — Round 2, March 2019. https://classic.mceliece.org/nist/mceliece-20190331.pdf.

BCS13. Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: fast constant-time code-based cryptography. In Guido Bertoni and Jean-Sébastien Coron, editors, *Cryptographic Hardware and Embedded System – CHES 2013*, volume 8086 of *LNCS*, pages 250–272. Springer, 2013.

BDH11. Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS – a practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography – PQCrypto 2011*, volume 7071 of *LNCS*, pages 117–129. Springer, 2011. Second Version, Cryptology ePrint Archive, Report 2011/484.

BJMM12. Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in 2n/20: How $1+1=0$ improves information set decoding. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 520–536. Springer, 2012.

BLP11. Daniel J. Bernstein, Tanja Lange, and Christiane Peters. Smaller decoding exponents: Ball-collision decoding. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *LNCS*, pages 743–760. Springer, 2011.

CD10. Pierre-Louis Cayrel and Pierre Dusart. McEliece/Niederreiter PKC: Sensitivity to fault injection. In *5th International Conference on Future Information Technology*, pages 1–6. IEEE, May 2010.

CEvMS16. Cong Chen, Thomas Eisenbarth, Ingo von Maurich, and Rainer Steinwandt. Horizontal and vertical side channel analysis of a mceliece cryptosystem. *IEEE Trans. Inf. Forensics Security*, 11(6):1093–1105, June 2016.

Dum89. I. I. Dumer. Two decoding algorithms for linear codes. *Probl. Peredachi Inf.*, 25:24–32, 1989.

FJP14. Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Mathematical Cryptology*, 8(3):209–247, 2014.

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

Gro96. Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Symposium on the Theory of Computing – STOC 1996*, pages 212–219. ACM, 1996.

HBG+18. Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. *RFC*, 8391:1–74, 2018.

HG13. Stefan Heyse and Tim Güneysu. Code-based cryptography on reconfigurable hardware: tweaking Niederreiter encryption for performance. *JCEN*, 3(1):29–43, 2013.

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

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

Mar17. Bernard Marr. 6 practical examples of how quantum computing will change our world, July 2017. Forbes.

McE78. Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report*, 42–44:114—116, 1978.

MMT11. Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 107–124. Springer, 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, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *LNCS*, pages 203–228. Springer, 2015.

MOP07. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer, 2007.

Nay19. Chris Nay. IBM unveils world's first integrated quantum computing system for commercial use, January 2019. IBM News Room.

Nie86. Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Probl. Control Inform.*, 15:19–34, 1986.

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

RHHM17. Mélissa Rossi, Mike Hamburg, Michael Hutter, and Mark E. Marson. A side-channel assisted cryptanalytic attack against QcBits. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *LNCS*, pages 3–23. Springer, 2017.

Sho94. Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Foundations of Computer Science – FOCS '94*, pages 124–134. IEEE, 1994.

Sho99. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

SS92. Vladimir M. Sidelnikov and Stepan O. Shestakov. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Math. Appl.*, 2(4):439–444, 1992.

SSMS10. Abdulhadi Shoufan, Falko Strenzke, H. Gregor Molter, and Marc Stöttinger. A timing attack against Patterson algorithm in the McEliece PKC. In Donghoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology – ICISC 2009*, volume 5984 of *LNCS*, pages 161–175. Springer, 2010.

SSPB19. Simona Samardjiska, Paolo Santini, Edoardo Persichetti, and Gustavo Banegas. A reaction attack against cryptosystems based on LRPC codes. In Peter Schwabe and Nicolas Thériault, editors, *Progress in Cryptology – LATINCRYPT 2019*, volume 11774 of *LNCS*, pages 197–216. Springer, 2019.

Ste89. Jacques Stern. A method for finding codewords of small weight. In Gérard Cohen and Jacques Wolfmann, editors, *Coding Theory and Applications*, volume 388 of *LNCS*, pages 106–113. Springer, 1989.

TE15. Mostafa Taha and Thomas Eisenbarth. Implementation attacks on post-quantum cryptographic schemes. Cryptology ePrint Archive, Report 2015/1083, 2015.

Wel47. Bernard Lewis Welch. The Generalization of 'Student's' Problem when Several Different Population Variances are Involved. *Biometrika*, 34(1/2):28–35, 1947.

WSN17. Wen Wang, Jakub Szefer, and Ruben Niederhagen. FPGA-based key generator for the Niederreiter cryptosystem using binary Goppa codes. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *LNCS*, pages 253–274. Springer, 2017.

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 – PQCrypto 2018*, volume 10786 of *LNCS*, pages 77–98. Springer, 2018.

A Appendix

First we state a lemma whose validity can be easily checked, but it will be useful in the proof of Proposition 1.

Lemma 1. *For $n, w, \beta, a \in \mathbf{N}$ the following identity holds:*

$$\prod_{i=0}^{a} \frac{\binom{n-w-i\beta}{\beta}}{\binom{n-i\beta}{\beta}} = \frac{\binom{n-(a+1)\beta}{w}}{\binom{n}{w}}.\square$$

A.1 Proof of Proposition 1

Proof. We denote by $H(n, i, \beta)$ the event of hitting a high chunk in the i-th query, after hitting low chunks in the first $i-1$ queries. Then the probability of the event $H(n, i, \beta)$ is:

$$\Pr(H(n,i,\beta)) = \frac{\binom{n-w}{\beta}}{\binom{n}{\beta}} \cdot \dots \cdot \frac{\binom{n-(i-2)\beta-w}{\beta}}{\binom{n-(i-2)\beta}{\beta}} \cdot (1 - \frac{\binom{n-(i-1)\beta-w}{\beta}}{\binom{n-(i-1)\beta}{\beta}}),$$

which by Lemma 1 is

$$\Pr(H(n,i,\beta)) = \frac{\binom{n-(i-1)\beta}{w}}{\binom{n}{w}} - \frac{\binom{n-i\beta}{w}}{\binom{n}{w}}.$$

Plugging in these values in the expectation

$$E(\beta) = \sum_{i \geqslant 0} i \cdot \Pr(H(n,i,\beta)),$$

we obtain equation (4).

For the second part of the Proposition, let $\Pr(w(\beta) = j|High)$ denote the probability that the weight of the chunk is j under the condition that it is a high chunk. Then,

$$\Pr(w(\beta) = j|High) = \frac{\Pr(w(\beta) = j)}{\Pr(High)} = \frac{\Pr(w(\beta) = j)}{1 - \Pr(Low)},$$

where Low is the event of a low chunk.

Since $\Pr(w(\beta) = j) = \frac{\binom{w'}{j}\binom{n'-w'}{\beta-j}}{\binom{n'}{\beta}}$ and $\Pr(Low) = \frac{\binom{n'-w'}{\beta}}{\binom{n'}{\beta}}$, we obtain

$$\Pr(w(\beta) = j|High) = \frac{\binom{w'}{j}\binom{n'-w'}{\beta-j}}{\binom{n'}{\beta} - \binom{n'-w'}{\beta}}.$$

Now

$$E(w(\beta)) = \sum_{j=1}^{\beta} j \cdot \Pr(w(\beta) = j|High) = \frac{1}{\binom{n'}{\beta} - \binom{n'-w'}{\beta}} \sum_{j=1}^{\beta} j \cdot \binom{w'}{j}\binom{n'-w'}{\beta-j} =$$

$$= \frac{1}{\binom{n'}{\beta} - \binom{n'-w'}{\beta}} \sum_{j=1}^{\beta} w' \cdot \binom{w'-1}{j-1}\binom{n'-w'}{\beta-j} = \frac{w'}{\binom{n'}{\beta} - \binom{n'-w'}{\beta}}\binom{n'-1}{\beta-1}.$$