Side Channel Information Set Decoding using Iterative Chunking

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

Abstract. This paper presents an attack based on side-channel information and information set decoding (ISD) on the code-based Niederreiter cryptosystem and an evaluation of the practicality of the attack using an electromagnetic side channel. We start by directly adapting the timing side-channel plaintext-recovery attack by Shoufan et al. from 2010 to the constant-time implementation of the Niederreiter cryptosystem as used in the official FPGA-implementation of the NIST finalist "Classic McEliece". We then enhance our attack using ISD and a new technique that we call *iterative chunking* to further significantly reduce the number of required side-channel measurements. We theoretically show that our attack improvements have a significant impact on reducing the number of required side-channel measurements. For example, for the 256-bit security parameter set `kem/mceliece6960119` of "Classic McEliece", we improve the basic attack that requires 5415 measurements to less than 562 measurements on average to mount a successful plaintext-recovery attack. Further reductions can be achieved at the price of increasing the cost of the ISD computations. We confirm our findings by practically mounting the attack on the official FPGA-implementation of "Classic McEliece" for all proposed parameter sets.

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

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,

This work has been funded by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE and by the European Commission through the ERC Starting Grant 805031 (EPOQUE).

and weather forecast [24]. 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 [29]. Even larger experimental quantum computers are operating in the labs of Google, IBM, and Microsoft. However, besides the high hopes on 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 solve the integer factorization and discrete logarithm problems in polynomial time using Shor's quantum-computer algorithm [35], 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 revisiting alternative cryptographic schemes that are able to withstand attacks by quantum computers. The most popular approaches are multivariate, hash-, lattice-, code-, and isogeny-based cryptography. For details on the basic ideas behind these approaches, we refer the reader to, e.g., [5, 15]. Code-based cryptography is often regarded as the most mature and reliable, but with a major drawback of being much less efficient than, e.g., lattice-based cryptosystems. The McEliece [27] and the Niederreiter [30] cryptosystems using binary Goppa codes are typically considered as conservative but safe post-quantum solutions.

The National Institute of Standards and Technology (NIST) started a public process for the standardization of PQC schemes [12] in November 2017; schemes from all classes mentioned above have been submitted. Very recently, the standardization process entered its third and final phase. The "Classic McEliece" cryptosystem [6] was chosen as one of the four finalists for the standardization of key-encapsulation mechanisms (KEMs) [1]. It is highly expected by the community that "Classic McEliece" will become part of a NIST standard of PQC.

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., 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 like power consumption and electromagnetic, photonic, or acoustic emissions. 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 [40].

In this work, we focus on the "Classic McEliece" cryptosystem, the well understood and trusted KEM finalist in the NIST standardization process, and describe a plaintext-recovery attack on its decryption algorithm using side-channel information. The "Classic McEliece" cryptosystem — though honouring

Robert J. McEliece, the pioneer of code-based cryptography, with its name — is using the equivalent approach proposed by Harald Niederreiter as described in Section 2.3.

Our Contributions. We start by directly adapting the side-channel attack from Shoufan et al. [36] for plaintext recovery on the McEliece cryptosystem to a side-channel attack on the Niederreiter cryptosystem. There are some important differences that we overcome in this adaptation: The attack of Shoufan et al. is aiming at a timing side channel present due to the non-constant time Patterson's decoder; in contrast, we attack the constant-time hardware reference implementation [41] of "Classic McEliece" that uses a constant-time implementation of the Berlekamp-Massey (BM) decoder. Therefore, a timing side channel does not exist any more — instead we perform an electromagnetic (EM) side-channel attack.

Our attack is a reaction-based attack that makes use of decoding failures: Adding more errors to the ciphertext leads to a failed decoding of the BM decoder and the output error-locator polynomial has very few roots. This can be detected over the EM side channel and used to learn the value of the error position.

Our main contribution is the optimization of the number of required side-channel queries in our reaction-based attack:

1. We introduce a new technique that we call *iterative chunking* which enables us to iteratively increase the number of learned error positions (chunks) in one (cumulative) query. We analyze our approach and theoretically derive an estimate for the optimal chunk size for an attack based on the system parameters. Our technique provides huge improvement in the number of required queries of up to 90% for the "Classic McEliece" parameters [7].
2. We further improve our attack by introducing the possibility of a trade-off between required queries and computational power. We do this by performing a certain amount of queries, reducing the problem to a smaller one, and applying known information set decoding algorithms on the remaining problem. The trade-off strongly depends on the computational capabilities of the attacker, but even for relatively small computational effort of 2^{40} operations, we can further reduce the necessary queries by around 15%.

We implement and demonstrate a practical attack on the official hardware implementation [41] of "Classic McEliece" [7]. The practically achieved improvements almost perfectly match our theoretical analysis.

Related Work. In [36] Shoufan et al. present a timing attack on the McEliece cryptosystem that recovers the plaintext of a given ciphertext using a decryption oracle (see Section 2.4). 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 [10]. A differential power analysis (DPA) attack is presented in [11] that recovers the secret key of a QC-MDPC McEliece FPGA implementation by measuring the leakage of the carry

occurring during the key rotation operation. A similar attack on a software implementation is presented in [15], using the detection of counter overflows. An attack described in [33] uses information gained by DPA about the positions of set bits to recover the secret key in a cryptanalytic attack.

The attack in [36] by Shoufan et al. can be considered as a reaction-based side channel attack. In a different scenario, reaction attacks have been successfully applied to several code-based cryptosystems [18, 17, 34, 2]. In these attacks, the attacker (typically) sends carefully chosen encrypted messages to a decryption oracle and observes whether these cause decryption failures. Based only on observing whether there was a failure, these attacks can extract the secret key.

Information set decoding is a well known decoding technique that is dating back to the work of Prange [31] in the 1960's. The basic approach has been improved throughout the years by the works of Lee and Brickell [21], Leon [22], Stern [39] (and concurrently Dumer [14]) who first proposed to use *collision decoding* (actually the term was introduced later [9]). All subsequent improvements build on top of Stern's algorithm by exploring more refined techniques for collision search. The list is extensive and includes: [16, 9, 25, 3, 26].

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

Structure of this paper. Section 2 provides some background information on information set decoding and on the code-based McEliece and Niederreiter cryptosystems and gives a brief introduction to the side-channel attack from Shoufan et al. [36]. Section 3 follows up with a description of our adaption of Shoufan et al.'s attack to Niederreiter and our improvements for reducing the number of queries with iterative chunking. Here, we mathematically estimate the optimal parameter for our chunking strategy, describe an implementation using an ideal decryption oracle, and discuss the first evaluation results. In Section 4, we provide a leakage analysis of the FPGA implementation from [41] using EM leakage, present a construction of a practical decryption oracle, and evaluate the entire approach practically. We discuss the applicability of the iterative chunking approach in Section 5 and conclude the paper in Section 6.

Notation. In the following, $\mathrm{GF}(q)$ denotes the Galois field of order q. Capital bold letters like $\mathbf{H}$ denote matrices and small bold letters denote column vectors over a Galois field, e.g., plaintext message $\mathbf{m}$, ciphertext $\mathbf{c}$, and error vector $\mathbf{e}$. The corresponding rows vectors are denoted as $\mathbf{m}^\top$. The i-th column vector of a matrix $\mathbf{H}$ is denoted as $\mathbf{H}_i$ and the j-th coordinate in a vector $\mathbf{m}$ as $\mathbf{m}_j$. The function $\mathrm{w}(\mathbf{v})$ returns the Hamming weight (HW) of an input vector $\mathbf{v}$.

2 Background

In this section, we briefly introduce information set decoding, the McEliece cryptosystem, and its dual variant, the Niederreiter cryptosystem, which will be the object of our attack, as well as the timing attack from Shoufan et al. [36].

2.1 Information Set Decoding

Suppose we are given a parity check matrix $\mathbf{H} \in \mathrm{GF}(2)^{(n-k)\times n}$ of a binary $[n, k]$ code of dimension k and length n, and a syndrome $\mathbf{s} \in \mathrm{GF}(2)^{n-k}$. An information set decoding (ISD) algorithm solves the decoding problem:

$$\textit{Find } \mathbf{e} \in \mathrm{GF}(2)^n, \mathrm{w}(\mathbf{e}) = w \textit{ such that } \mathbf{H} \cdot \mathbf{e} = \mathbf{s}. \tag{1}$$

Basically, the algorithm guesses the error vector on k coordinates, and then uses this information to obtain the remaining error coordinates. The set of k coordinates is called *information set*, since it carries enough information to recover the entire error vector. The decoding problem gives rise to a linear system with the error coordinates $\mathbf{e}_1, \ldots, \mathbf{e}_n$ as unknowns. If k coordinates are correctly guessed, the system 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.

ISD was proposed by Prange [31]. 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 [39] 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) [25] and the Becker-Joux-May-Meurer (BJMM) [3] 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 [26] 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.

2.2 McEliece Cryptosystem

In 1978, McEliece proposed a cryptosystem using error correcting codes [27]. The basic idea of this cryptosystem is to use an error correcting code with 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 $\mathbf{G}$. Encryption works by computing a code word for the plaintext $\mathbf{m}$ using the generator matrix and by adding an error $\mathbf{e}$ with $\mathrm{w}(\mathbf{e}) \leq t$ that is small enough so that the error correction algorithm is able to correct the error. The ciphertext $\mathbf{c}$ is therefore computed as $\mathbf{c}^\top = \mathbf{m}^\top \mathbf{G} + \mathbf{e}^\top$. The receiver simply corrects the error by applying his secret error correction algorithm and recovers the plaintext from the code word. The security of the system is based on the hardness of decoding a general linear code, a problem known to be NP-hard [4], and the difficulty to recover the secret structure of the code from the public generator matrix.

2.3 Niederreiter Cryptosystem

In 1986, Niederreiter proposed a dual variant of the McEliece cryptosystem using a $(n-k) \times n$ parity-check matrix $\mathbf{H}$ instead of a generator matrix as public key [30]. In this case, an error vector $\mathbf{e}$ of weight $\mathrm{w}(\mathbf{e}) = t$ is the plaintext; the syndrome $\mathbf{s} = \mathbf{He}$ 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 with 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 [37]. Today, there are many variants 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 [6] and Round 2 [7] (see also [8]). We have summarized the notation that we will use in Table 1.

Key generation of the Niederreiter cryptosystem using binary Goppa codes works as follows (see [41]): Choose a random irreducible polynomial $g(x)$ over $\mathrm{GF}(2^m)$ of degree t and a list $(\alpha_0, \alpha_1, \ldots \alpha_{n-1}) \in \mathrm{GF}(2^m)^n$ of distinct elements of $\mathrm{GF}(2^m)$ (the support). From $g(x)$ and $(\alpha_0, \alpha_1, \ldots \alpha_{n-1})$, compute the $t \times n$ matrix $\tilde{\mathbf{H}}$ over $\mathrm{GF}(2^m)$. Transform $\tilde{\mathbf{H}}$ into a $mt \times n$ binary matrix $\mathbf{H}$ by replacing each $\mathrm{GF}(2^m)$-entry by a m-bit column. Finally, compute the systematic form $\mathbf{H}' = [\mathbf{I}_{mt}|\mathbf{K}]$ of $\mathbf{H}$ (where $\mathbf{I}_{mt} \in \mathrm{GF}(2)^{mt \times mt}$ denotes the identity matrix) and return $g(x)$ and $(\alpha_0, \alpha_1, \ldots \alpha_{n-1})$ as private key and $\mathbf{K}$ as public key. The last step of computing the systematic form $\mathbf{H}'$ of the parity-check matrix $\mathbf{H}$ compresses the size of the public key from mtn bits to $mt(n-mt)$ bits, because the preceding identity matrix $\mathbf{I}_{mt}$ does not need to be stored or communicated.

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

Decryption of the syndrome depends on the error-correcting algorithm used. Examples are Patterson's algorithm and the BM algorithm. Using the BM algorithm as in [8, 41], decryption works as follows: First, use the idea attributed to Sendrier in [19] and compute the double-size $2t \times n$ matrix $\tilde{\mathbf{H}}^{(2)}$ over $\mathrm{GF}(2^m)$.

Symbol	Description
$m \in \mathbb{N}$	Size of the binary field.
$t \in \mathbb{N}$	Correctable errors.
$n \in \mathbb{N}$	Code length.
$k \in \mathbb{N}$	Code dimension ($k = n - mt$).
$g(x)$	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 $\mathrm{GF}(2^m)$.
$\mathbf{H} \in \mathrm{GF}(2)^{mt \times n}$	Parity-check matrix.
$\mathbf{H}' = [\mathbf{I}_{mt} \vert \mathbf{K}] \in \mathrm{GF}(2)^{mt \times n}$	Parity-check matrix in systematic form.
$\mathbf{K} \in \mathrm{GF}(2)^{mt \times (n-mt)}$	Public key.
$\mathbf{e} \in \mathrm{GF}(2)^n$	Error vector (plaintext).
$\mathbf{s} \in \mathrm{GF}(2)^{mt}$	Syndrome (ciphertext).
$\sigma(x)$	Error-locator polynomial over $\mathrm{GF}(2^m)$ of degree t.

Table 1. Symbols for "Classic McEliece" (Niederreiter) [7, 41].

	`kem/mceliece-`				
	`348864`	`460896`	`6688128`	`6960119`	`8192128`
m	12	13	13	13	13
t	64	96	128	119	128
n	3488	4608	6688	6960	8192
$k = n - mt$	2720	3360	5024	5413	6528

Table 2. Parameter sets of "Classic McEliece" [7].

Compute the double-size syndrome $\mathbf{s}^{(2)} = \tilde{\mathbf{H}}^{(2)} \cdot (\mathbf{s}|0)$ by appending $n - mt$ zeros to the syndrome $\mathbf{s}$. Now, we can use the BM algorithm to compute the error-locator polynomial $\sigma(x)$ of $\mathbf{s}^{(2)}$. 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 i-th bit of the error vector $\mathbf{e}_i = 1$, otherwise $\mathbf{e}_i = 0$.

A KEM is constructed in "Classic McEliece" from the basic encryption/decryption primitives using a standard transformation. Table 2 shows the parameters proposed by [7].

2.4 Timing Side-Channel Attack on McEliece

Shoufan et al. in [36] describe a plaintext-recovery attack on the McEliece cryptosystem 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 in 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 Reaction-based Side-Channel Analysis

In this section, we describe our reaction-based plaintext-recovery attack on the Niederreiter cryptosystem. In Section 3.1, we explain how to adapt the timing attack by Shoufan et al. [36] introduced in Section 2.4 on the McEliece cryptosystem to an EM side-channel attack on the Niederreiter cryptosystem. We describe how to reduce the number of queries required for a side-channel attack when using ISD in Section 3.2 and we improve our basic attack in Section 3.3 using the *iterative chunking* technique. Further, we mathematically estimate the optimal parameter for our query strategy and evaluate its implementation with a simulation using an ideal decryption oracle. Finally, we explain how to combine the ISD techniques with our improved attack in Section 3.4.

3.1 Side-Channel Attack on Niederreiter

In the attack by Shoufan et al. in [36] 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. Here, the equivalent of adding an error to a code word in McEliece, is to add a column of the parity-check matrix (i.e., the public key) to the syndrome (i.e., the ciphertext). Therefore, the attack from [36] can trivially be adapted 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

Algorithm 1: Iterative Reaction-based SCA

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

1 $\mathbf{e} \leftarrow (0, \ldots, 0)$;
2 **for** $i \in E = \{1, \ldots, n\}$ **do**
3 $\quad \mathbf{s}' \leftarrow \mathbf{s} \oplus \mathbf{H}'_i$;
4 $\quad$ **if** Oracle($\mathbf{s}'$) $= true$ **then** $\mathbf{e}_i \leftarrow 1$;
5 **end**
6 **return** $\mathbf{e}$;

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 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 Patterson's algorithm is used, a timing side-channel attack as in [36] can be mounted on Niederreiter as well.

Attacking constant-time implementations. Modern implementations typically avoid timing side channels by providing a constant-time implementation of critical algorithms. Thus, in this case another side channel is required to mount the attack. In Section 4, we investigate the EM side channel in the reference hardware implementation by Wang et al. [41] using a constant-time implementation of 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.

For a side-channel attack based on EM, the attacker needs to be in possession of the device under attack, e.g., a smart card or a security token, that has physical measures protecting secret information such as private and secret keys, but no explicit countermeasures prohibiting the exploitation of the side channel. Furthermore, the attacker needs to be in possession of a ciphertext that he intends to decrypt, e.g., intercepted on a communication channel. Under these requirements, the attacker can perform a series of measurements of EM emissions of the device under attack while decrypting manipulated ciphertexts.

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 ranges from 3488 to 8192 queries for the NIST parameters of "Classic

Algorithm 2: ISD-supported Iterative Reaction-based SCA

input : Same as Algorithm 1
output: Same as Algorithm 1

1 $\widehat{E} \subset E = \{1, \ldots, n\}, |\widehat{E}| \leqslant k$;
2 Same as Algorithm 1 lines **1-5** with $\widehat{E}$ instead of E
3 $\tilde{\mathbf{s}} \leftarrow \mathbf{s} - \mathbf{H}' \cdot \mathbf{e}$;
4 $\tilde{\mathbf{e}} := (\mathbf{e}_i)_{i \in E \setminus \widehat{E}}$; $\hat{\mathbf{e}} := (\mathbf{e}_i)_{i \in \widehat{E}}$;
5 $\tilde{\mathbf{H}}' := (\mathbf{H}'_i)_{i \in E \setminus \widehat{E}}$;
6 $\tilde{\mathbf{e}} \leftarrow \mathrm{ISD}(n - |\widehat{E}|, k - |\widehat{E}|, w - \mathrm{w}(\hat{\mathbf{e}}), \tilde{\mathbf{H}}', \tilde{\mathbf{s}})$;
7 $\mathbf{e} \leftarrow \mathrm{Reconstruct}(\hat{\mathbf{e}}, \tilde{\mathbf{e}})$;
8 **return** $\mathbf{e}$;

McEliece" (cf. Section 2.3). 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.

3.2 Reducing the Number of Queries with Information Set Decoding

In the reaction attack described in Algorithm 1, we clearly do not have to reconstruct the entire error vector (all error coordinates) using side-channel information. We can recover an information set of size $k < n$, instead, and use basic linear algebra to recover the rest of the error vector (cf. Section 2.1). We can do even better if the attacker is in a position to trade-off queries for computational power — first collect a number of queries *less* than k, use them to reduce the problem to a smaller one, and then solve the smaller problem using some of the ISD algorithms described in Section 2.1.

In more detail, let $\mathrm{ISD}(n, k, w, \mathbf{H}, \mathbf{s})$ be any ISD algorithm, such as Stern's or Ball Collision decoding, that on input of a parity check matrix $\mathbf{H} \in \mathrm{GF}(2)^{(n-k) \times n}$ and syndrome $\mathbf{s} \in \mathrm{GF}(2)^{n-k}$ outputs an error vector $\mathbf{e} \in \mathrm{GF}(2)^n$ with $\mathrm{w}(\mathbf{e}) = w$ — 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 $\mathbf{e}_i$ of the error vector. Using the oracle, we learn a subset of error indices $\widehat{E} \subset E = \{1, \ldots, n\}$ where $|\widehat{E}| \leqslant k$. We denote the corresponding subvector of $\mathbf{e}$ by $\hat{\mathbf{e}} = (\mathbf{e}_i)_{i \in \widehat{E}}$ and its complement by $\tilde{\mathbf{e}} = (\mathbf{e}_i)_{i \in E \setminus \widehat{E}}$. Similarly $\widehat{\mathbf{H}} = (\mathbf{H}'_i)_{i \in \widehat{E}}$ and $\tilde{\mathbf{H}}' = (\mathbf{H}'_i)_{i \in E \setminus \widehat{E}}$. From the obtained information, setting $\tilde{\mathbf{s}} = s - \widehat{\mathbf{H}} \cdot \hat{\mathbf{e}}$, the decoding problem (1) transforms to:

$$\tilde{\mathbf{H}}' \cdot \tilde{\mathbf{e}} = \tilde{\mathbf{s}}. \tag{2}$$

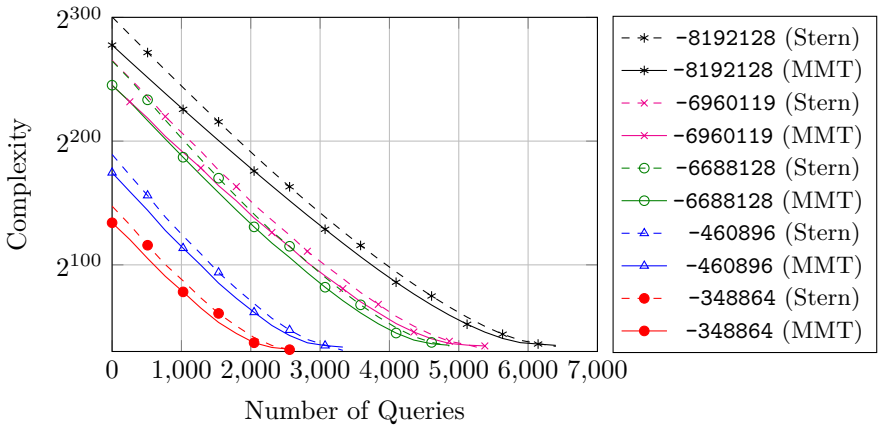

Fig. 1. Time-queries trade-off when using ISD decoding algorithms.

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

The performance of Algorithm 2 depends directly on the size of the 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 he performs based on his 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.

3.3 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 the number of errors has not increased (instead of reduced as in Section 3.1) 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%.

Iterative chunking with chunks of size $\beta = 2$. Suppose that instead of a single error index, we query two error indices (a chunk of size $\beta = 2$) at once. We

$(\mathbf{e}_i, \mathbf{e}_j)$	$(\mathbf{e}'_i, \mathbf{e}'_j)$	$w(\mathbf{e}')$	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 3. All cases of the response of the decryption oracle when querying chunks of size two at once ($\mathbf{s}' = \mathbf{s} \oplus \mathbf{H}'_i \oplus \mathbf{H}'_j$). The first column shows the initial state of the queried chunk, the second shows the state of the pair after 'flipping' the values, the third shows the total number of errors in the new state, and the last column shows the oracle's answer.

first randomly select a chunk (i, j) of error indices, $i, j \in \{1, \ldots, n\}, i \neq j$. We add both columns $\mathbf{H}'_i$ and $\mathbf{H}'_j$ to the syndrome $\mathbf{s}$ to obtain the new syndrome $\mathbf{s}'$. We give the input $\mathbf{s}'$ to the decryption oracle. Note that the decryption oracle will output *false* only in the case when the values at the corresponding error indices in the error vector were $(\mathbf{e}_i, \mathbf{e}_j) = (0, 0)$ (which we call 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 $(\mathbf{e}_i, \mathbf{e}_j) = (0, 0)$, after adding the pair of columns $(\mathbf{H}'_i, \mathbf{H}'_j)$ to the syndrome, we obtain $(\mathbf{e}'_i, \mathbf{e}'_j) = (1, 1)$, and in total $w + 2$ errors. Hence, the number of errors has increased beyond w and the decryption oracle will output *false*. If $(\mathbf{e}_i, \mathbf{e}_j) = (0, 1)$ or $(\mathbf{e}_i, \mathbf{e}_j) = (1, 0)$, we get $(\mathbf{e}'_i, \mathbf{e}'_j) = (1, 0)$ and $(\mathbf{e}'_i, \mathbf{e}'_j) = (0, 1)$ 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, $(\mathbf{e}_i, \mathbf{e}_j) = (1, 1)$, after adding the columns we obtain $(\mathbf{e}'_i, \mathbf{e}'_j) = (0, 0)$. So in this case, the number of errors reduces to $w - 2$, and the decryption oracle returns *true* as well. Table 3 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 $(\mathbf{e}_i, \mathbf{e}_j) = (0, 0)$. 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.

The improvement using chunks of two is easy to see: Since the length of the error vector is much bigger than its Hamming weight, most of the time the randomly chosen chunk will be $(\mathbf{e}_i, \mathbf{e}_j) = (0, 0)$, and we can confirm these values by *only one* query, instead of two as in the approach from the previous section.

Iterative chunking for $\beta > 2$. This simple strategy for $\beta = 2$ is already significantly better than the naïve approach from the previous section, but we can do much better by extending this idea to chunks $(\mathbf{e}_{i_1}, \ldots, \mathbf{e}_{i_\beta})$ 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.

		$\mathbf{s}$		$\mathbf{s}' = \mathbf{s} \oplus \mathbf{H}'_i$	
$(\mathbf{e}_i, \mathbf{e}_j, \mathbf{e}_k)$	$(\mathbf{e}'_i, \mathbf{e}'_j, \mathbf{e}'_k)$	$w(\mathbf{e}')$	Oracle	$w(\mathbf{e}') - 1$	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 4. Overview of the oracle answers for $\beta = 3$ when the number of errors in the syndrome $\mathbf{s}$ are reduced to $w(\mathbf{e}') - 1$ ($\mathbf{s}' = \mathbf{s} \oplus \mathbf{H}'_i$) with the knowledge of an error position i in the original error vector $\mathbf{e}$.

First, note that we cannot directly use the same approach for chunks of size $\beta > 2$. For example for $\beta = 3$ we have Table 4 analogous to Table 3. Table 4 shows (columns 3 and 4) that there is ambiguity in the oracle answers, so if the oracle answers *false* we cannot distinguish whether the chunk was $(0,0,0)$, $(0,0,1)$, $(0,1,0)$, or $(1,0,0)$. 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 4 show. This requires knowledge about the position of one 1 in the error vector. Adding the corresponding column of the matrix $\mathbf{H}'$ to the syndrome reduces the number of errors to $w-1$. So how can we find the position of one 1? Well, this can easily be done by first querying chunks of size $\beta = 2$ until a 'high' chunk is found. Querying both positions within the 'high' chunk of size $\beta = 2$ reveals the position of one 1. 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. To summarize, the procedure informally goes as follows:

Part I: For a chunk size β, starting at $\beta = 2$:

1. Query random chunks of size β without replacement until the oracle returns *true* which indicates a 'high' chunk.
2. Inspect the positions within the 'high' chunk and locate the 1s.
3. Use these 1s to increase the size of the chunks that we query: by adding to the syndrome a column $\mathbf{H}'_i$ of the matrix $\mathbf{H}'$ corresponding to a 1 at position i in the error vector we reduce the number of errors by one.
4. Increase the chunk size to $\beta + 1$ and repeat from Step 1 until $\beta = \beta_T$.

Part II: When a threshold β_T is reached, change the procedure to:

1. Query random chunks of size β_T without replacement. If the oracle returns *true* save the 'high' chunk in a bucket of capacity $n-k$.
2. End the whole procedure when k error positions have been learned.

The threshold β_T is an optimization parameter — the optimal value for the chunk size β at which we need to stop increasing. We determine its value so that the number of necessary queries to recover an information set is minimized.

Note that, in Part II, we only care about finding enough 'low' chunks so that we recover an information set. So in principle, we can throw away the 'high' chunks, unless there are too few chunks remaining — not enough to recover an information set. This is why we save them in a bucket. After the bucket has been filled, we start inspecting 'high' chunks as well, because we will not be able to recover an information set otherwise.

Remark 1. Typically there will be only one 1 in a 'high' chunk, so to simplify our analysis, in Part I we will always increase the size of the queried chunks only by one, although in theory, it is also possible to increase by more than one, precisely by the weight of the 'high' chunk.

Remark 2. We could continue Part I, i.e., increase the size of the queried chunks, until we have learned k error positions — enough to recover an information set. However, the increase only makes sense as long as there is a good probability that the queried chunk is 'low' — in which case one can learn β zeros in the error vector in only one query. In contrast, if the chunk size is 'high', since we want to increase the chunk size further, a more expensive inspection with additional queries is required (around $\mathcal{O}(\log \beta)$ queries using a divide-and-conquer strategy). Thus, if the chunk is 'high' with big probability, the advantage from the increased chunk size quickly diminishes.

The details of the procedure are given in Algorithm 3. The algorithm for inspecting the 'high' chunks is given in Algorithm 4. It uses a divide-and-conquer strategy to reduce the number of needed queries.

Finding the optimal $\boldsymbol{\beta_T}$. 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 us make some simple but important observations.

The process of querying chunks of size β can be modeled as a sequence of independent and identical Bernoulli trials in which success means querying a 'high' chunk. Indeed, since we assume uniform distribution of the error vectors, the success probability of all the trials is the same, and depends only on the number of 1s in the error vector and the size of the queried chunks. As a result, the number of queries needed to find a 'high' chunk follows the geometric distribution. Based on these observations, we have the following results.

Proposition 1. *Suppose we query chunks of size β without replacement, until we find a 'high' chunk.*

a. *The probability that a queried chunk is high is* $p_\beta = 1 - \frac{\binom{n-w}{\beta}}{\binom{n}{\beta}}$.
b. *The probability that the weight of the chunk is j under the condition that it is a 'high' chunk is given by*

$$\Pr(w(\beta) = j | High) = \frac{\binom{w}{j}\binom{n-w}{\beta-j}}{\binom{n}{\beta} - \binom{n-w}{\beta}}. \tag{3}$$

Algorithm 3: Iterative Chunking with $\beta \geq 2$

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

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

$\mathbf{e}' \leftarrow [0, \ldots, 0]$; // initialize with zero vector
$\beta \leftarrow 2$; // start with chunk size 2
$\mathbf{s}'[1] \leftarrow \mathbf{s}$; $\mathbf{s}'[\beta] \leftarrow \mathbf{s}$; // list of syndromes $\mathbf{s}'[i]$ 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
  $\mathbf{s}'' \leftarrow \mathbf{s}'[\beta]$ + Sum([$\mathbf{H}'[i]$ ***for*** i ***in*** *chunk*]) ; // add columns in chunk to $\mathbf{s}'$
  if Oracle($\mathbf{s}''$) = *true* **then** // a 'high' chunk is found
    if $\beta < \beta_T$ **or** $|bucket| > n - k$ **then** // conditions for inspecting
      $e_{id} \leftarrow$ FindErrorPositions(*chunk*, $\mathbf{s}', \mathbf{H}'$); // inspect 'high' chunk
      for i **in** e_{id} **do**
        $\mathbf{e}_i \leftarrow 1$; // update $\mathbf{e}'$ with found errors from 'high' chunk
        if $\beta < \beta_T$ **then**
          $\mathbf{s}'[\beta + 1] \leftarrow \mathbf{s}'[\beta] - \mathbf{H}'[i]$; // remove found errors from $\mathbf{s}'$
          $\beta \leftarrow \beta + 1$; // increase chunk size, up to β_T
      end
    else
      bucket $\leftarrow$ *bucket* + *chunk* ; // collect chunks with remaining errors
end
return $\mathbf{e}'$, *bucket*, *indices*;

c. The expected number of queries until we find a 'high' chunk of size β is

$$E(\beta) = \frac{1}{p_\beta}. \tag{4}$$

d. The expected weight of a 'high' chunk of size β is

$$E(w(\beta)) = \frac{w\binom{n-1}{\beta-1}}{\binom{n}{\beta} - \binom{n-w}{\beta}}. \tag{5}$$

Proof. a. Directly, since the probability of hitting a low chunk is

$$\Pr(Low) = \frac{\binom{n-w}{\beta}}{\binom{n}{\beta}}.$$

Algorithm 4: FindErrorPositions

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

$stack \leftarrow [\mathrm{Left}(chunk), \mathrm{Right}(chunk)]$;
$e_{id} \leftarrow []$;
while *stack not empty* **do**
  $chunk \leftarrow \mathrm{Pop}(stack)$;
  $s'' \leftarrow s'[\mathrm{Len}(chunk)] + \mathrm{Sum}([\mathbf{H}'[i]$ ***for*** i ***in*** $chunk])$;
  if $\mathrm{Oracle}(s'') = true$ **then** $next \leftarrow chunk$; // there is an error position
  else $next \leftarrow \mathrm{Pop}(stack)$; // continue search in the stack head
  if $\mathrm{Len}(next) = 1$ **then** // found an error position
    $\mathrm{Push}(e_{id}, next[0])$;
    $stack \leftarrow [\mathrm{Flatten}(stack)]$; // inspect entire stack
  else // split the next chunk directly
    $\mathrm{Push}(stack, \mathrm{Left}(next))$;
    $\mathrm{Push}(stack, \mathrm{Right}(next))$;
end
return e_{id};

b. 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,

$$\Pr(w(\beta) = j|High) = \frac{\Pr(w(\beta) = j)}{p_\beta}.$$

Since $\Pr(w(\beta) = j) = \frac{\binom{w}{j}\binom{n-w}{\beta-j}}{\binom{n}{\beta}}$ we obtain (3).

c. Since the querying of chunks of size β follows the geometric distribution, we immediately obtain the claim.

d. From a. we can directly calculate

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

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

Now that we know the expected weight $E(w(\beta))$ of a 'high' chunk from Proposition 1, we should also estimate the number of queries to inspect a 'high' chunk, i.e., to determine all error positions within the 'high' chunk. Various strategies can be applied for this task. The simplest one is to just query all positions, which requires β queries, but this would make sense only if many 1s are expected within the chunk. In our problem we typically have a 'high' chunk of

weight only 1, so a divide-and-conquer approach is more suitable: We split the chunk in half and query the left half depth first. When one 1 is identified, we collect all remaining unknown positions (denoted as "Flatten(*stack*)" in Algorithm 4), and query them at once as one chunk. The probability is high that this chunk will be 'low', so we do not need any more queries. If it happens that the chunk is high, we repeat the same procedure for this smaller chunk. The details are given in Algorithm 4. Proposition 2 estimates the number of queries needed to inspect a 'high' chunk using this algorithm.

Proposition 2. *Suppose we inspect a 'high' chunk of size β. Then, the expected number of queries to learn all positions within the 'high' chunk of size β using Algorithm 4 is bounded from above by*

$$E_{High}(\beta) \leqslant \sum_{j=1}^{\beta} \left(\frac{\binom{w}{j}\binom{n-w}{\beta-j}}{\binom{n}{\beta} - \binom{n-w}{\beta}}\right) \cdot \left(j - \frac{j}{\beta} + \sum_{i=0}^{j-1} \log_2(\beta - i)\right). \tag{6}$$

Proof. To estimate $E_{High}(\beta)$, we will first write it as

$$E_{High}(\beta) = \sum_{j=1}^{\beta} \Pr(w(\beta) = j | High) \cdot E_{High}(\beta | w(\beta) = j),$$

where $E_{High}(\beta|w(\beta) = j)$ denotes the expected number of queries to learn all positions within a high chunk of size β under the condition that there are exactly j 1s in the chunk.

We consider first $E_{High}(\beta|w(\beta) = 1)$. Note that we need $\log_2 \beta$ queries to locate the 1, and on average $1 - 1/\beta$ additional queries for the remaining unqueried positions. Here, $-1/\beta$ comes from the case where the 1 is on the last β-th position of the chunk. Hence $E_{High}(\beta|w(\beta) = 1) = \log_2 \beta + 1 - 1/\beta$. Next,

$$\begin{aligned}
&E_{High}(\beta|w(\beta) = 2) = \\
&= \frac{1}{\binom{\beta}{2}} \sum_{\substack{1 \leqslant i,j \leqslant \beta \\ j \neq \beta}} (\log_2 \beta + \log_2(\beta - i) + 2) + \frac{1}{\binom{\beta}{2}} \sum_{\substack{1 \leqslant i,j \leqslant \beta \\ j = \beta}} (\log_2 \beta + \log_2(\beta - i) + 1) = \\
&= \frac{1}{\binom{\beta}{2}} \sum_{1 \leqslant i,j \leqslant \beta} (\log_2 \beta + \log_2(\beta - i)) + 2 - \frac{\binom{\beta-1}{1}}{\binom{\beta}{2}} \leqslant \\
&\leqslant \frac{1}{\binom{\beta}{2}} \sum_{1 \leqslant i,j \leqslant \beta} (\log_2 \beta + \log_2(\beta - 1)) + 2 - \frac{2}{\beta} = \log_2 \beta + \log_2(\beta - 1) + 2 - \frac{2}{\beta}
\end{aligned}$$

where the second sum in the first row comes from the fact that if the second 1 is on the last position, we need one less query. We bound $E_{High}(\beta|w(\beta) = 2)$ from above, instead of calculating it exactly, in order to simplify the expression and the analysis. It is a rather tight bound, which can be confirmed by the simulation and experiments we have performed (see Table 5).

Using induction it is easy to show that

$$E_{High}(\beta|w(\beta)=j)=\sum_{i=0}^{j-1}\log_2(\beta-i)+j-\frac{j}{\beta}\ .$$

Finally, $\Pr(w(\beta)=j|High)$ can be easily calculated from Proposition 1, which gives the final expression. □

Using Propositions 1 and 2 we can now estimate the number of queries required in Algorithm 3.

Proposition 3. *The number of queries required to recover k error positions using Algorithm 3 is given by*

$$Q(\beta_T)=\sum_{i=2}^{\beta_T-1}(E(i)+E_{High}(i))+N_1\cdot E(\beta_T)+N_2\cdot(E(\beta_T)+E_{High}(\beta_T)),\quad(7)$$

where N_1 and N_2 are given by

$$N_1=min\{\frac{n-k}{\beta_T},\frac{k-\sum_{i=2}^{\beta_T-1}i\cdot E(i)}{\beta_T\cdot(E(\beta_T)-1)}\},\ N_2=\frac{k-\sum_{i=2}^{\beta_T-1}i\cdot E(i)+N_1\cdot\beta_T}{\beta_T\cdot E(\beta_T)}-N_1.\quad(8)$$

The optimal β_T is then the one that minimizes the number of queries $Q(\beta_T)$.

Proof. With the notation introduced so far, the number of error positions $I_1(\beta_T)$ that we learn before we reach the threshold β_T and the required queries $Q_1(\beta_T)$ in the process is

$$I_1(\beta_T)=\sum_{i=2}^{\beta_T-1}i\cdot E(i),\quad\text{and}\quad Q_1(\beta_T)=\sum_{i=2}^{\beta_T-1}(E(i)+E_{High}(i)).\qquad(9)$$

When we reach the threshold value β_T, we change the strategy and continue to query only chunks of size β_T. Recall that we do not inspect 'high' chunks, but save them in a bucket of capacity $n-k$ and only start inspecting them if the bucket is full and we have not yet recovered k error positions. Suppose we query $N_1\cdot E(\beta_T)$ chunks while the bucket is sill not full and $N_2\cdot E(\beta_T)$ chunks after the bucket is full, for some unknown N_1,N_2. The number of learned error positions $I_2(\beta_T)$ and the required number of queries $Q_2(\beta_T)$ is

$$\begin{aligned}I_2(\beta_T)\ &=N_1\cdot\beta_T\cdot(E(\beta_T)-1)+N_2\cdot\beta_T\cdot E(\beta_T),\\ Q_2(\beta_T)&=N_1\cdot E(\beta_T)+N_2\cdot(E(\beta_T)+E_{High}(\beta_T)).\end{aligned}\qquad(10)$$

The algorithm stops when k error positions have been learned, i.e., when the condition

$$I_1(\beta_T)+I_2(\beta_T)\geqslant k\qquad(11)$$

is satisfied. Since the bucket has capacity $n-k$, we also have the condition

$$N_1\cdot\beta_T\leqslant n-k.\qquad(12)$$

From (11) and (12) we find the expressions in (8) for N_1 and N_2. Now, (7) follows by combining (9) and (10). □

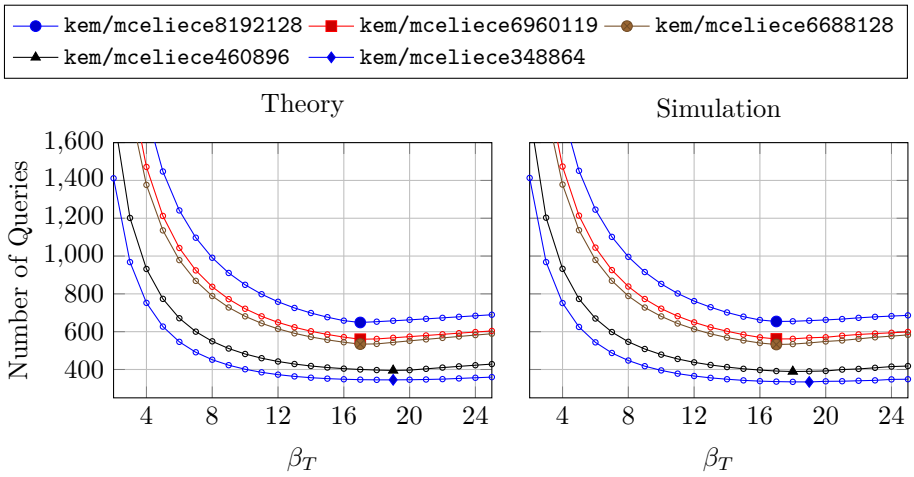

Fig. 2. The expected number of queries needed for threshold values from 2 to 25 both for the theoretical prediction and averaged simulations. The minima are marked.

Evaluation. Using the estimate from Proposition 3 we get the expected number of queries for $\beta_T \in \{2, \ldots, 25\}$ for all the parameter sets `kem/mceliece348864`, `kem/mceliece460896`, `kem/mceliece6688128`, `kem/mceliece6960119`, as well as `kem/mceliece8192128` of "Classic McEliece" (see Table 2).

In order to evaluate our findings, we implemented the described iterative chunking strategy using Python and SageMath[3] and ran the implementation as a simulation with an ideal oracle that always returns the correct response. The simulation was applied to ten different key pairs using ten different plaintext/ciphertext pairs for each key pair (100 in total) per parameter set and β_T value. We generated the key pairs as well as the plain- and ciphertext pairs using the SageMath scripts that are enclosed with the publicly available FPGA implementation of the Niederreiter cryptosystem from Wang et al.

The optimal threshold values for the parameter sets together with the number of needed queries are given in Table 5 and depicted in Figure 2. The results of the simulation and the later experiments in Section 4.3 show that the estimate matches very well (see Figure 2). For the parameter set `kem/mceliece460896`, in the simulation $\beta_T = 18$ turned out to be slightly more efficient than the expected $\beta_T = 19$. However, we believe that this is only because the number of queries for $\beta_T = 18$ and $\beta_T = 19$ in the theoretical estimate are very close to each other.

Compared to recovering the information set without iterative chunking (the case of aquiring k traces from the oracle in Algorithm 2) the number of traces is decreased by approximately 87.7% for `kem/mceliece348864` and 90% for `kem/mceliece8192128`.

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

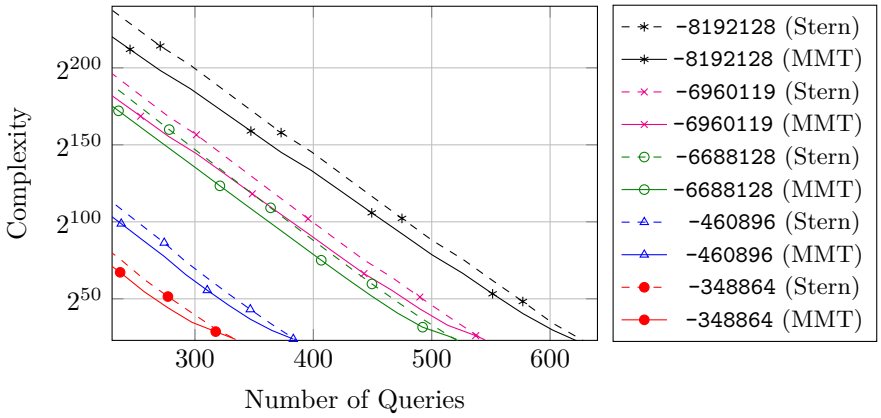

Fig. 3. Time-queries trade-off when using optimal iterative chunking and ISD.

3.4 Combining Iterative Chunking with Information Set Decoding

The number of needed queries can be further decreased by combining iterative chunking with some non-trivial ISD algorithm. Instead of recovering an entire information set from queries, we can stop early, when we have learned only $\delta < k$ 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 at 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.

In order to express the trade-off more accurately, but at the same time avoid complicated expressions due to in-between or corner cases, we will discretize the possible values for the number of performed queries and learned error positions. This discretization is quite natural, since it follows exactly the steps of our algorithm. Using the notation from the proof of Proposition 3 we consider the partial sums of the number of performed queries $Q(\beta_T)$ defined as

$$Q_i(\beta_T) = \begin{cases} Q_{i-1}(\beta_T) + E(i) + E_{High}(i), & i \in \{2, \ldots, \beta_T - 1\} \\ Q_{i-1}(\beta_T) + E(\beta_T) + b \cdot E_{High}(\beta_T), & i \in \{\beta_T, \ldots, \beta_T + N_1 + N_2 - 1\} \end{cases}$$

with $Q_1(\beta_T) = 0$ and $b = 0$ if $i < \beta_T + N_1$ and $b = 1$ otherwise.

Similarly, we define the partial sums of the learned error positions $I(\beta_T)$ as

$$I_i(\beta_T) = \begin{cases} I_{i-1}(\beta_T) + iE(i), & i \in \{2, \ldots, \beta_T - 1\} \\ I_{i-1}(\beta_T) + \beta_T(E(\beta_T) - (1-b)), & i \in \{\beta_T, \ldots, \beta_T + N_1 + N_2 - 1\} \end{cases}$$

with $I_1(\beta_T) = 0$ and $b = 0$ if $i < \beta_T + N_1$ and $b = 1$ otherwise. Let $I'_i(\beta_T)$ be the same as $I_i(\beta_T)$ except $b = 1$ for all $i \in \{\beta_T, \ldots, N_1+N_2-1\}$. Finally, let $w_1 = 0$ and $w_i = w_{i-1} + E(i)$ for $i \in \{2, \ldots, \beta_T + N_1+N_2-1\}$.

Now it is not difficult to verify that the trade-off between performed queries and computational power is given by the following proposition.

Proposition 4. *An attacker performing $Q_i(\beta_T)$ queries to the decryption oracle can recover the secret message by solving the $ISD(n - I'_i(\beta_T), k - I_i(\beta_T), w - w_i)$ problem. The case of $i = 1$ corresponds to the case of no side-channel information, and the case of $i = \beta_T + N_1+N_2-1$ corresponds to recovering an entire information set using side-channel information.*

4 Attack Evaluation

For the practical attack evaluation we adapted the implementation presented in [41] for field programmable gate arrays (FPGAs). In Section 4.1 we describe our approach for a preliminary leakage analysis of the decryption module design and in Section 4.2 we construct 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 [41] in detail to find a proper point of interest at which we can find significant leakage. The selected implementation uses a constant-time implementation of the BM algorithm for the error correction. The BM algorithm returns an error-locator polynomial that 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 [41] 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 VCD-file (value change dump) of an Icarus Verilog (iverilog) simulation using a simple Hamming-distance model. This results in a

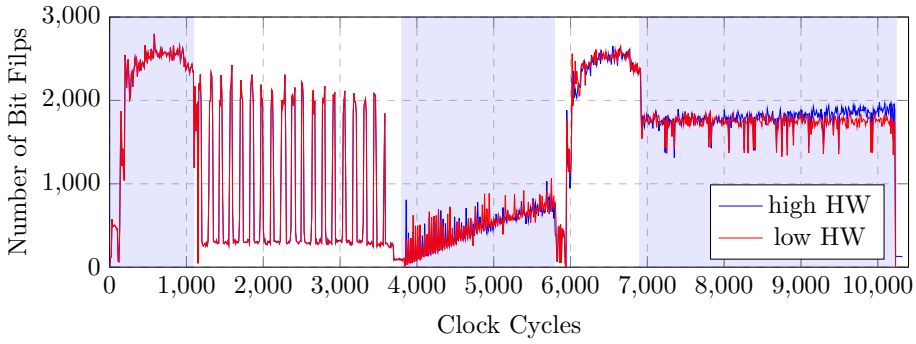

Fig. 4. Simulated power consumption based on a Hamming-distance model for successful decoding (high HW in the resulting error vector) and unsuccessful decoding (low HW) using a small parameter set with $m = 12$, $t = 66$, and $n = 3307$. The different parts of the algorithm can clearly be seen (marked 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.

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 may be leaked is at the last round of the second additive FFT operation that evaluates the error-locator polynomial returned 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 should be distinguishable in general. However, because the implementation of the additive FFT uses several multiplicator instances in parallel, the logical noise added to the exploitable leakage is quite high.

The second point of possible leakage 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.

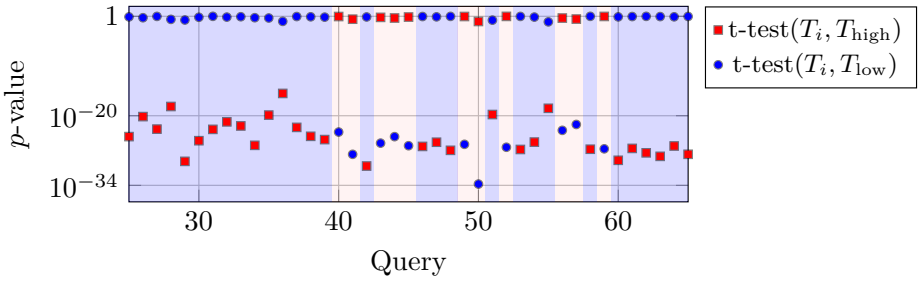

Fig. 5. Example p-values of the t-tests of consecutive oracle queries. For each query a trace T_i is compared to a known faulty (T_{low}) and a known decodable trace (T_{high}). If a trace is similar to the faulty trace (i.e., due to a higher p-value), a decoding failure is detected (light blue background).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

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.

In order to get a response for individual queried syndromes, we apply Welch's t-test [42] 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 $\mathbf{H}'$. The probability that this results in a 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 compute the difference p_Δ 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} as

$$p_\Delta = \text{t-test}(T_i, T_{high}) - \text{t-test}(T_i, T_{low}). \tag{13}$$

Thus, if the difference of the p-values p_Δ is positive, the acquired trace is similar to the original syndrome trace and interpreted as decodable. Otherwise, if p_Δ 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

Algorithm 5: GetReferences

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

Decrypt$(s) \rightsquigarrow T_{high}, Clk_{high}$;
$T'_{high} \leftarrow$ Compress(T_{high}, Clk_{high});
$\mathbf{s}^* \leftarrow \mathbf{s}$;
repeat
  for $i \leftarrow 1$ **to** 5 **do** $\mathbf{s}^* \leftarrow \mathbf{s}^* \oplus \mathbf{H}'_j, \quad j \in^R \{0 \dots n-1\}$;
  Decrypt$(\mathbf{s}^*) \rightsquigarrow T_{low}, Clk_{low}$;
  $T'_{low} \leftarrow$ Compress(T_{low}, Clk_{low});
until t-test$(T'_{high}, T'_{low}) \approx 0$;
return T'_{high}, T'_{low};

Algorithm 6: DEMA-based Oracle

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

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

decryption oracle therefore is

$$response = \begin{cases} true \text{ (decodable)}, & \text{if } p_\Delta > 0 \\ false \text{ (not decodable)}, & \text{otherwise} \end{cases} . \tag{14}$$

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 a trace compression similar as described in [23]. We reduce the raw signal to the maximum peak-to-peak difference of the amplitudes of the EM signal in each first clock half-wave and take it as the new value for the entire clock cycle. We just need to know the clock frequency to identify the clock cycle ranges in the raw signal.

Optimization. The quality metric of the side-channel oracle is the difference of the p-values p_Δ. The greater the absolute value is the better is the differentiability of the 'low' and the 'high' case. The more errors are removed from the syndrome during the iterative chunking process the less 1s are in the resulting

error vector and its weight decreases. Since we are using the Hamming weight of the error-vector construction as the exploitable leakage, $|p_\Delta|$ decreases during the attack as well. Therefore, we optimized our side-channel oracle by updating the reference trace T_{high} by the trace which is recorded if a single 1 is recovered.

4.3 Practical Evaluation

To evaluate the attack in practice, we ported the FPGA design of [41] to a Xilinx Kintex-7 (XC7K160T) on a SAKURA-X[4] 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 of a decryption operation. We acquired the EMR profiles and the trigger signal using a PicoScope 5244D MSO 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.

We analyzed the design of the Niederreiter decryption for all parameter sets proposed for "Classic McEliece" in the second round of the NIST PQC competition (see Table 2). Corresponding to the simulated trace in Figure 4, the five individual parts of the decryption process were identifiable.

We evaluated the ISD-supported iterative approach (Section 3.2) and the chunk-based approach (Section 3.3) using the oracle described in Algorithm 6 for all 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 $n - k$ columns that are collected in the bucket. If an attacker is able to invest additional computing power, he is able to reduce the number of traces even further, trading in a higher complexity of the ISD.

We are using the same implementation of the iterative chunking as the simulation described in Section 3.3 — however, we substituted the ideal oracle with the real side-channel oracle described in Algorithm 6. We examined the same data sets as used for the simulation, i.e., ten different key pairs using ten different plaintext/ciphertext pairs for each key pair (100 in total) per parameter set. To demonstrate the feasibility of the side-channel-based oracle, we ran the practical evaluation of the plaintext-recovery attack for the optimal chunk size threshold (β_T) that was determined from the simulation (see Table 5).

For each "Classic McEliece" parameter set we were able to recover the entire plaintexts with our iterative chunking approach. Since the responses of the side-channel oracle and the ideal oracle in the simulation are equal, we get the same average number of traces for the same value of β_T in the simulation as well as the experiments.

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

kem/ mceliece-	Approach	Simulation			Theory		Experiment
		min.	avg.	max.	plain	cost $\approx 2^{40}$	
348864	$\beta = 1$	–	2722 $(k+2)$	–			
	$\beta_T = 17$	281	335.89	483	346.17	—	—
	$\beta_T = 18$	275	334.51	481	345.16	—	—
	$\boldsymbol{\beta_T = 19}$	273	**334.16**	481	**345.06**	**287.26**	**334.16**
	$\beta_T = 20$	279	337.36	494	345.74	—	—
	$\beta_T = 21$	282	337.92	482	347.09	—	—
460896	$\beta = 1$	–	3362 $(k+2)$	–			
	$\beta_T = 16$	353	397.01	523	404.41	—	—
	$\beta_T = 17$	343	391.72	513	400.00	—	—
	$\boldsymbol{\beta_T = 18}$	337	**389.33**	514	396.95	—	**389.33**
	$\boldsymbol{\beta_T = 19}$	333	390.33	515	**395.06**	**337.66**	—
	$\beta_T = 20$	329	392.14	517	396.62	—	—
	$\beta_T = 21$	326	399.59	532	403.45	—	—
6688128	$\beta = 1$	–	5026 $(k+2)$	–			
	$\beta_T = 15$	505	553.56	608	556.73	—	—
	$\beta_T = 16$	480	540.10	598	544.22	—	—
	$\boldsymbol{\beta_T = 17}$	468	**532.11**	595	**534.14**	**470.91**	**532.11**
	$\beta_T = 18$	456	534.30	604	535.31	—	—
	$\beta_T = 19$	448	540.46	617	543.39	—	—
6960119	$\beta = 1$	–	5415 $(k+2)$	–			
	$\beta_T = 15$	529	584.15	708	585.21	—	—
	$\beta_T = 16$	518	569.93	692	571.24	—	—
	$\boldsymbol{\beta_T = 17}$	505	**561.29**	680	**559.87**	**493.90**	**561.29**
	$\beta_T = 18$	508	562.15	679	561.61	—	—
	$\beta_T = 19$	499	567.68	685	567.58	—	—
8192128	$\beta = 1$	–	6530 $(k+2)$	–			
	$\beta_T = 15$	618	679.41	788	676.59	—	—
	$\beta_T = 16$	596	661.87	771	658.28	—	—
	$\boldsymbol{\beta_T = 17}$	587	**653.97**	760	**648.66**	**576.96**	**653.97**
	$\beta_T = 18$	571	654.86	760	652.85	—	—
	$\beta_T = 19$	586	658.33	778	657.53	—	—

Table 5. Statistical data for the required number of queries for a successful recovery of an entire error vector. The data for the simulation and the experiments was gathered from the same sets of ten different key pairs and ten different plaintext/ciphertexts per key pair (100 in total) for each parameter set. We also give the theoretical prediction and the number of required queries for applying ISD at a cost of around 2^{40} operations. For comparison, we also give the number of traces when no chunking is used, i.e., $\beta = 1$ for all queries.

5 Application Perspectives of Iterative Chunking

In general, our iterative chunking approach requires a) a controllable decryption failure using public information (the public key) to manipulate the syndrome and b) a reliable decryption oracle for the feedback. The feedback can be, e.g., some side-channel information or an explicit protocol response. Thus, any decryption implementation, hardware or software, with these two characteristics is vulnerable to our iterative chunking approach if a reliable feedback can be established.

As presented in this work, the Berlekamp-Massey decoding algorithm exhibits the first required property. If also the second property, i.e., a side channel as feedback exists, then the following implementations of "Classic McEliece" are vulnerable to our attack: The NIST reference C implementation [7] which is based on the work of "McBits revisited" [13] and the implementation found in the project PQClean [20] which is included in several projects, e.g., the library from the Open Quantum Safe project [38]. The code-based scheme HQC [28] also uses the Berlekamp-Massey decoder and is attackable as well if a detectable difference is revealed when the decoding fails.

Furthermore, the attack of Shoufan et al. in [36] can be improved using iterative chunking when exploiting the distinguishable behavior of the deployed Patterson's decoder since it shows a timing side-channel information when the syndrome exceeds its decoding capacity.

Further open research questions are whether iterative chunking is applicable to rank-based schemes or if it can improve reaction attacks revealing the private key, as for example shown for CCA-secure lattice-based KEM schemes in [32].

6 Conclusion

In this paper, we showed that side-channel attacks on code-based cryptosystems can significantly be improved using our new iterative chunking approach and that the cost of an ISD attack can be significantly reduced when it is combined with this improved side-channel attack, revealing the plaintext of a given message.

If "Classic McEliece" is used as a basis for a KEM scheme in a key exchange, the plaintext is used to derive a session key even if the long-term decryption key is protected. Therefore, we suggest to research proper countermeasures against decryption leakage.

References

[1] G. Alagic, J. Alperin-Sheriff, D. Apon, D. Cooper, Q. Dang, J. Kelsey, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone. *NISTIR 8309: Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*. Tech. rep. National Institute of Standards and Technology, 2020.

[2] N. Aragon and P. Gaborit. "A key recovery attack against LRPC using decryption failures". In: *International Workshop on Coding and Cryptography – WCC 2019*. 2019.

[3] A. Becker, A. Joux, A. May, and A. Meurer. "Decoding Random Binary Linear Codes in 2n/20: How $1 + 1 = 0$ Improves Information Set Decoding". In: *Advances in Cryptology – EUROCRYPT 2012*. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. LNCS. Springer, 2012, pp. 520–536.

[4] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. "On the inherent intractability of certain coding problems (Corresp.)" In: *IEEE Trans. Inf. Theory* 24.3 (1978), pp. 384–386.

[5] D. J. Bernstein, J. Buchmann, and E. Dahmen, eds. *Post-Quantum Cryptography*. Springer, 2009. ISBN: 978-3-540-88702-7.

[6] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, and W. Wang. *Classic McEliece: conservative code-based cryptography — Round 1*. `https://classic.mceliece.org/nist/mceliece-20171129.pdf`. Nov. 2017.

[7] D. J. Bernstein, T. Chou, T. Lange, I. von Maurich, R. Misoczki, R. Niederhagen, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, and W. Wang. *Classic McEliece: conservative code-based cryptography — Round 2*. `https://classic.mceliece.org/nist/mceliece-20190331.pdf`. Mar. 2019.

[8] D. J. Bernstein, T. Chou, and P. Schwabe. "McBits: fast constant-time code-based cryptography". In: *Cryptographic Hardware and Embedded System – CHES 2013*. Ed. by G. Bertoni and J.-S. Coron. Vol. 8086. LNCS. Springer, 2013, pp. 250–272.

[9] D. J. Bernstein, T. Lange, and C. Peters. "Smaller Decoding Exponents: Ball-Collision Decoding". In: *Advances in Cryptology – CRYPTO 2011*. Ed. by P. Rogaway. Vol. 6841. LNCS. Springer, 2011, pp. 743–760.

[10] P.-L. Cayrel and P. Dusart. "McEliece/Niederreiter PKC: Sensitivity to Fault Injection". In: *5th International Conference on Future Information Technology*. IEEE, 2010, pp. 1–6.

[11] C. Chen, T. Eisenbarth, I. von Maurich, and R. Steinwandt. "Horizontal and Vertical Side Channel Analysis of a McEliece Cryptosystem". In: *IEEE Trans. Inform. Forensics Security* 11.6 (2016), pp. 1093–1105.

[12] L. Chen, D. Moody, and Y.-K. Liu. *Post-Quantum Cryptography*. NIST, `https://csrc.nist.gov/Projects/post-quantum-cryptography`.

[13] T. Chou. "McBits Revisited". In: *Cryptographic Hardware and Embedded System – CHES 2017*. Ed. by W. Fischer and N. Homma. Vol. 10529. LNCS. Springer, 2017, pp. 213–231.

[14] I. I. Dumer. "Two Decoding Algorithms for Linear Codes". In: *Probl. Peredachi Inf.* 25 (1 1989), pp. 24–32.

[15] L. D. Feo, D. Jao, and J. Plût. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies". In: *J. Mathematical Cryptology* 8.3 (2014), pp. 209–247.

[16] M. Finiasz and N. Sendrier. "Security Bounds for the Design of Code-Based Cryptosystems". In: *Advances in Cryptology – ASIACRYPT 2009*. Ed. by M. Matsui. Vol. 5912. LNCS. Springer, 2009, pp. 88–105.

[17] Q. Guo, T. Johansson, and P. Stankovski. "A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors". In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by J. H. Cheon and T. Takagi. Vol. 10031. LNCS. Springer, 2016, pp. 789–815.

[18] C. Hall, I. Goldberg, and B. Schneier. "Reaction Attacks against Several Public-Key Cryptosystem". In: *Information and Communication Security – ICICS 1999*. Ed. by V. Varadharajan and Y. Mu. Vol. 1726. LNCS. Springer, 1999, pp. 2–12.

[19] S. Heyse and T. Güneysu. "Code-based cryptography on reconfigurable hardware: tweaking Niederreiter encryption for performance". In: *JCEN* 3.1 (2013), pp. 29–43.

[20] M. J. Kannwischer, J. Rijneveld, P. Schwabe, D. Stebila, and T. Wiggers. *The PQClean Project*. `https://github.com/PQClean/PQClean`. Aug. 2020.

[21] P. J. Lee and E. F. Brickell. "An Observation on the Security of McEliece's Public-Key Cryptosystem". In: *Advances in Cryptology - EUROCRYPT 1988*. Ed. by C. G. Günther. Vol. 330. LNCS. Springer, 1988, pp. 275–280.

[22] J. S. Leon. "A probabilistic algorithm for computing minimum weights of large error-correcting codes". In: *IEEE Trans. Information Theory* 34.5 (1988), pp. 1354–1359.

[23] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer, 2007. ISBN: 978-0-387-30857-9.

[24] B. Marr. *6 Practical Examples Of How Quantum Computing Will Change Our World*. Forbes, `https://www.forbes.com/sites/bernardmarr/2017/07/10/6-practical-examples-of-how-quantum-computing-will-change-our-world/`. 2017.

[25] A. May, A. Meurer, and E. Thomae. "Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$". In: *Advances in Cryptology – ASIACRYPT 2011*. Ed. by D. H. Lee and X. Wang. Vol. 7073. LNCS. Springer, 2011, pp. 107–124.

[26] A. May and I. Ozerov. "On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes". In: *Advances in Cryptology – EUROCRYPT 2015*. Ed. by E. Oswald and M. Fischlin. Vol. 9056. LNCS. Springer, 2015, pp. 203–228.

[27] R. J. McEliece. "A Public-Key Cryptosystem Based On Algebraic Coding Theory". In: *DSN Progress Report* 42–44 (1978), pp. 114–116.

[28] C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J. Bos, J.-C. Deneuville, P. Gaborit, E. Persichetti, J.-M. Robert, P. Véron, and G. Zémor. *Hamming Quasi-Cyclic (HQC) — Second round version*. `http://pqc-hqc.org/doc/hqc-specification_2020-05-29.pdf`. May 2020.

[29] C. Nay. *IBM Unveils World's First Integrated Quantum Computing System for Commercial Use*. IBM News Roomm `https://newsroom.ibm.`

com/2019-01-08-IBM-Unveils-Worlds-First-Integrated-Quantum-Computing-System-for-Commercial-Use. 2019.

[30] H. Niederreiter. "Knapsack-type cryptosystems and algebraic coding theory". In: *Probl. Control Inform.* 15 (1986), pp. 19–34.

[31] E. Prange. "The use of information sets in decoding cyclic codes". In: *IRE Trans. Information Theory* 8.5 (1962), pp. 5–9.

[32] P. Ravi, S. Sinha Roy, A. Chattopadhyay, and S. Bhasin. "Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems – TCHES* 2020.3 (2020), pp. 307–335.

[33] M. Rossi, M. Hamburg, M. Hutter, and M. E. Marson. "A Side-Channel Assisted Cryptanalytic Attack Against QcBits". In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by W. Fischer and N. Homma. Vol. 10529. LNCS. Springer, 2017, pp. 3–23.

[34] S. Samardjiska, P. Santini, E. Persichetti, and G. Banegas. "A Reaction Attack Against Cryptosystems Based on LRPC Codes". In: *Progress in Cryptology – LATINCRYPT 2019*. Ed. by P. Schwabe and N. Thériault. Vol. 11774. LNCS. Springer, 2019, pp. 197–216.

[35] P. W. Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[36] A. Shoufan, F. Strenzke, H. G. Molter, and M. Stöttinger. "A Timing Attack against Patterson Algorithm in the McEliece PKC". In: *Information, Security and Cryptology – ICISC 2009*. Ed. by D. Lee and S. Hong. Vol. 5984. LNCS. Springer, 2010, pp. 161–175.

[37] V. M. Sidelnikov and S. O. Shestakov. "On insecurity of cryptosystems based on generalized Reed-Solomon codes". In: *Discrete Math. Appl.* 2.4 (1992), pp. 439–444.

[38] D. Stebila and M. Mosca. *Open Quantum Safe Project*. https://github.com/open-quantum-safe/liboqs. Aug. 2020.

[39] J. Stern. "A method for finding codewords of small weight". In: *Coding Theory and Applications*. Ed. by G. Cohen and J. Wolfmann. Vol. 388. LNCS. Springer, 1989, pp. 106–113.

[40] M. Taha and T. Eisenbarth. *Implementation Attacks on Post-Quantum Cryptographic Schemes*. Cryptology ePrint Archive, Report 2015/1083.

[41] W. Wang, J. Szefer, and R. Niederhagen. "FPGA-Based Niederreiter Cryptosystem Using Binary Goppa Codes". In: *Post-Quantum Cryptography – PQCrypto 2018*. Ed. by T. Lange and R. Steinwandt. Vol. 10786. LNCS. Springer, 2018, pp. 77–98.

[42] B. L. Welch. "The Generalization of 'Student's' Problem when Several Different Population Variances are Involved". In: *Biometrika* 34.1/2 (1947), pp. 28–35.