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 Niederreiter cryptosystem and an evaluation of the practicality of the attack using an electromagnetic side channel. First, we describe a basic plaintext-recovery attack on the decryption algorithm of the Niederreiter cryptosystem. In case the cryptosystem is used as Key-Encapsulation Mechanism (KEM) in a key exchange, the plaintext corresponds to a session key. 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 decoding algorithm to the Niederreiter cryptosystem using the constant time Berlekamp-Massey decoding algorithm. We then enhance our attack by utilizing an ISD approach to support the basic attack and we introduce iterative column 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. 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. For example, for the 256bit-security parameter set `kem/mceliece6960119` we improve the basic attack that requires 5415 measurements to on average of about 560 measurements to mount a successful plaintext recovery attack. Further reductions can be achieved at increasing cost of the ISD computations.

Keywords: ISD · Reaction Attack · 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, and weather forecast [21]. 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 [25]. 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 [30, 31], 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 hash-, lattice-, code-, and isogeny-based as well as multivariate cryptography [4, 13]. Hash-based cryptography is used for very reliable signature schemes. 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 [24] and the Niederreiter [26] 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 [11]; 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., 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 [35].

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 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 Niederreiter as described in Section 2.3.

Our Contributions. We show how to adapt the side-channel attack from Shoufan et al. in [32] for plaintext recovery on the McEliece cryptosystem to a side-channel attack on the Niederreiter cryptosystem. The attack of Shoufan et al. is aiming at a timing side channel due to the non-constant time Patterson's decoder, while we perform an EM side-channel attack on the „Classic McEliece" hardware reference implementation [37], which is using a constant time Berlekamp-Massey decoder. If the Niederreiter cryptosystem is used as basis for a KEM in a key exchange, the plaintext constitutes the shared secret that is used to derive a session key. Thus, not just only a single plaintext message can be obtained by the attacker but all following encrypted communication in the session is revealed.

We optimize the number of required side-channel queries using a reaction-based attack combined with a technique that we call *iterative chunking.* The approach enables us to iteratively increase the number of learned error positions (chunks) in one (cumulative) query. We theoretically estimate the optimal chunk size for an attack based on the system parameters. We further improve our attack by applying known Information Set Decoding (ISD) algorithms and analyze the computational cost in relation to the number of required side-channel queries. While using ISD techniques reduces the required number of queries, it comes at a cost of computational power, so the trade-off strongly depends on the computational capabilities of the attacker.

We investigate the hardware implementation by Wang et al. in [37] accompanying the NIST submission "Classic McEliece" [5] for possible information leakage and we demonstrate the feasibility of our attack exploiting an electromagnetic field (EM) side channel in that hardware implementation.

Related Work. In [32] 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 [9]. A Differential Power Analysis (DPA) attack is presented in [10] that recovers the secret key of a QC-MDPC McEliece 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 [13], using the detection of counter overflows. An attack described in [28] uses information gained by DPA about the positions of set bits to recover the secret key in a cryptanalytic attack.

The attack in [32] can be considered as a sort of SCA reaction attack. In a different scenario, reaction attacks have been successfully applied to several code based cryptosystems [16, 15, 29, 1]. 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 [27] in the 1960's. The basic approach has been improved throughout the years by the works of Lee and Brickell [18], Leon [19], Stern [34] (and concurrently Dumer [12]) who first proposed to use *collision decoding* (actually the term was introduced later [8]). All subsequent improvements build on top of Stern's algorithm by exploring more refined techniques for collision search. The list is extensive and includes: [14, 8, 22, 2, 23].

We are not aware of a previous work that combines information set decoding with side channel analysis and 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. [32]. Section 3 follows up with a description of our adaption of that attack to Niederreiter and our improvements for reducing the number of queries with iterative chunking. Here, we mathematically predict the optimal parameter for our chunking strategy, describe an implementation of it using an ideal decryption oracle, and discuss first evaluation results. In Section 4, we provide a leakage analysis of the FPGA implementation from [37] using EM leakage, present a construction of a practical decryption oracle, and evaluate the entire approach practically. 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). $\mathrm{GF}(q)$ denotes the Galois field of order q.

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. [32].

2.1 Information Set Decoding

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

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

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. The decoding problem gives rise to the system:

$$H \cdot e = s \tag{2}$$

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 [27]. 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 [34] 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) [22] and the Becker-Joux-May-Meurer (BJMM) [2] 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 [23] 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 [24]. 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 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. The security of the system is based on the hardness of decoding a general linear code, a problem known to be NP-hard [3].

2.3 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 [26]. 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 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 [33]. 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 [5] and Round 2 [6] (see also [7]). 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 (following the notation in [36]): 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 $\mathrm{GF}(2^m)$. From $g(x)$ and $(\alpha_0, \alpha_1, \ldots \alpha_{n-1})$, compute the $t \times n$ parity-check matrix H 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 $(\alpha_0, \alpha_1, \ldots \alpha_{n-1})$ 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 Patterson's algorithm and the constant-time Berlekamp-Massey (BM) algorithm. Using the BM algorithm as in [7, 37], decryption works as follows: Since the BM algorithm can only correct up to $t/2$ errors, use the trick attributed to Sendrier in [17] and compute the double-size $2t \times n$ parity-check matrix $H^{(2)}$ 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 $e_i = 0$.

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.
$(\alpha_0, \alpha_1, \dots \alpha_{n-1}) \in \mathrm{GF}(2^m)^n$	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 for Classic McEliece (Niederreiter) [6, 37].

			`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 [6].

A KEM is constructed in “Classic McEliece” from the basic encryption/decryption primitives using a standard transformation. Table 2 shows the parameters proposed by [6].

2.4 Timing Side-Channel Attack on McEliece

Shoufan et al. describe a plaintext recovery attack on the McEliece cryptosystem in [32] 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 Reaction-based Side-Channel Analysis

In this section, we describe our reaction-based plaintext-recovery attack on the Niederreiter cryptosystem. We explain how we adapt the timing attack by Shoufan et al. [32] introduced in Section 2.4 on the McEliece cryptosystem to an EM side-channel attack on the Niederreiter cryptosystem in Section 3.1. 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 larger chunks in each query. Further, we mathematically predict the optimal parameter for our query startegy 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 [32] 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 [32] 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 a reduced number of errors from the cases with an increased number of errors, a

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 $e \leftarrow (0, \ldots, 0)$;
2 **for** $i \leftarrow 0$ **to** $n-1$ **do**
3 $\quad s' \leftarrow s \oplus H''[i]$;
4 $\quad$ **if** Oracle$(s') = \mathit{true}$ **then** $e[i] \leftarrow 1$;
5 **end**
6 **return** e;

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 [32] can be mounted on Niederreiter as well.

Attacking constant time implementations. In modern implementations, often a constant time algorithm, typically the Berlekamp-Massey (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 [37] 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.

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 she 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 is a few thousand for practical Niederreiter parameters (see 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

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 Oracle$(s') = \mathit{true}$ **then** $e[i] \leftarrow 1$;
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 attacker to significantly reduce the number of decoding operations that she needs to query from the device under attack.

3.2 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 ranges from 3488 to 8192 queries for the NIST parameters of "Classic McEliece" given in Table 2. 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 2.1). The information set can be recovered from obtained queries or using some of the ISD algorithms described in Section 2.1. 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

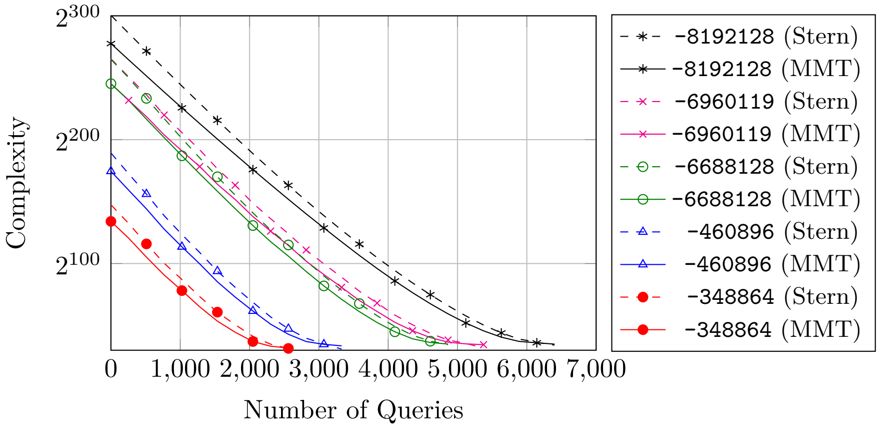

Fig. 1. Time-queries trade-off when using ISD decoding algorithms.

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 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 she performs based on her 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

(e_i, e_j)	(e'_i, e'_j)	$w(e')$	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 ($s' = s \oplus h_i \oplus h_j$). 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's answer.

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 $\beta = 2$. Suppose that instead of a single error index, we query two error indices (a chunk of size $\beta = 2$) at once. We first randomly select the chunk (i, j) of error indices, $i, j \in \{1, \dots, n\}, i \neq j$. 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 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 $(e_i, e_j) = (0, 0)$. 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

		s		$s' = s \oplus h_i$	
(e_i, e_j, e_k)	(e'_i, e'_j, e'_k)	$w(e')$	Oracle	$w(e') - 1$	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 s are reduced to $w(e') - 1$ ($s' = s \oplus h_i$) with the knowledge of an error position i in the original error vector e.

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.

Iterative chunking for $\beta > 2$. This simple strategy for $\beta = 2$ is already much better than the naïve approach from the previous paragraph, but we can do much better by extending this 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 cannot directly use the same approach for chunks of size $\beta \geqslant 2$. 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 we can not 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 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: We start with chunk size 2 and continue iteratively. For a chunk size β, we query random chunks without replacement until the oracle returns *true* which indicates a 'high' chunk. Next, we inspect the positions within the 'high' chunk, and locate the 1s. Now, we can use these 1s to increase the size of the chunks that we query: by adding to the syndrome a column h_i of the matrix H'' corresponding to a 1 at position i in the error vector we reduce the number of errors by one. This enables us to increase the size of the queried chunks by one as well. Typically there will be only one 1 in a 'high' chunk, so to simplify our analysis 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.

Let $E(\beta)$ be the expected number of queries on chunks of size β until we find a 'high' chunk (including the query on the 'high' chunk). Then after this part, we have learned $\beta \cdot E(\beta)$ error positions — which translates to the same number of new equations for our ISD problem.

Next, we increase the chunk size to $\beta+1$, and repeat the same procedure. We could continue increasing 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 divide-and-conquer strategy). Thus, if the chunk is 'high' with big probability, the advantage from the increased chunk size quickly diminishes. Hence, there exists an optimal value for the chunk size β at which we need to stop increasing. We wil call this threshold value 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 do not need to increase the chunk size anymore, it is not essential to find the position of a 1s in the error vector — 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, we can safely ignore the 'high' chunks, but instead of throwing them away, we save them in a bucket of capacity $n-k$. 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.

The whole procedure ends when we have recovered k error positions, i.e. when we have recovered an information set. The details are given in Algorithm 3. The algorithm for inspecting the 'high'chunks and locating the 1s within the chunk is given in Algorithm 4. It uses a divide and conquer strategy to reduce the number of needed queries.

Find the optimal β_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.

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

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 $\mathrm{Len}(indices) > \beta$ **do**
    $chunk \leftarrow \mathrm{Pop}(indices, \beta)$; // pop β-many indices
    $s'' \leftarrow s'[\beta] + \mathrm{Sum}([H''[i]$ ***for*** i ***in*** $chunk])$; // add columns in chunk to s'
    if $\mathrm{Oracle}(s'') = true$ **then** // there is an error position
        if $\beta < \beta_T$ **or** $|bucket| > n - k$ **then** // find errors until β_T is reached
            $e_{id} \leftarrow \mathrm{FindErrorPositions}(chunk, s', H'')$; // 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 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 e', *bucket*, *indices*;

Based on these observations, we have the following results. The proofs are given in Appendix A.

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

a. Let p_β denote the probability that the queried chunk is high.

$$p_\beta = 1 - \frac{\binom{n-w}{\beta}}{\binom{n}{\beta}}$$

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

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

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.

$stack \leftarrow [\mathrm{Left}(chunk), \mathrm{Right}(chunk)]$;
$e_{id} \leftarrow []$;
while *stack not empty* **do**
  $chunk \leftarrow \mathrm{Pop}(stack)$;
  $s'' \leftarrow s'[\mathrm{Len}(chunk)] + \mathrm{Sum}([H''[i]$ ***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};

c. *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,*

$$\Pr(w(\beta) = j|High) = \frac{\binom{w}{j}\binom{n-w}{\beta-j}}{\binom{n}{\beta} - \binom{n-w}{\beta}}. \tag{5}$$

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

$$E(w(\beta)) = \frac{w\binom{n-1}{\beta-1}}{\binom{n}{\beta} - \binom{n-w}{\beta}}. \tag{6}$$

Proposition 1 estimates the number of needed queries to find a 'high' chunk of size β as well as the expected weight of the found chunk. 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, which mainly depend on the expected weight of a 'high' chunk. 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 chance 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. The proof is given in Appendix A.

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

$$E_{High}(\beta) \leqslant \sum_{j=1}^{\beta} \Big(\frac{\binom{w}{j}\binom{n-w}{\beta-j}}{\binom{n}{\beta} - \binom{n-w}{\beta}}\Big) \cdot (j - \frac{j}{\beta} + \sum_{i=0}^{j-1} \log_2(\beta - i)). \tag{7}$$

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

Recall that Algorithm 3 first queries chunks of size $2, \ldots, \beta_T$ by iteratively increasing the chunk size right after a 'high' chunk has been found. As introduced earlier, let $E(\beta)$ denote the expected number of queries needed to find a 'high' chunk of size β. Then, the number of error positions that we learn at this step is $\beta E(\beta)$. Repeating the same for chunk sizes $2, 3 \ldots, \beta_T - 1$, we learn a total of

$$I_1(\beta_T) = \sum_{i=2}^{\beta_T - 1} i \cdot E(i) \tag{8}$$

error positions.

When we reach the threshold value β_T we change the strategy, and continue to query only chunks of size β_T. Since we do not need to increase the chunk size, we do not inspect 'high' chunks, but save them in a bucket of capacity $n - k$. Once the bucket is full we start also inspecting the 'high' chunks, since otherwise we will not be able to recover an information set (that is of size k). 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 is

$$I_2(\beta_T) = N_1 \cdot \beta_T \cdot (E(\beta_T) - 1) + N_2 \cdot \beta_T \cdot E(\beta_T) \tag{9}$$

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

$$I_1(\beta_T) + I_2(\beta_T) \geqslant k \tag{10}$$

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

$$N_1 \cdot \beta_T \leqslant n - k \tag{11}$$

Since the algorithm proceeds until the information set is found or the bucket is full, from equations (10) and (11) we find N_1 and N_2 as:

$$N_1 = min\{\frac{n-k}{\beta_T}, \frac{k - I_1(\beta_T)}{\beta_T \cdot (E(\beta_T) - 1)}\} \tag{12}$$

$$N_2 = \frac{I_2(\beta_T) - N_1 \cdot \beta_T \cdot (E(\beta_T) - 1)}{\beta_T \cdot E(\beta_T)} \tag{13}$$

Now it is easy to calculate the number of queries for a given threshold β_T. Where

$$Q_1(\beta_T) = \sum_{i=2}^{\beta_T-1} (E(i) + E_{High}(i)) \tag{14}$$

$$Q_2(\beta_T) = N_1 \cdot E(\beta_T) + N_2 \cdot (E(\beta_T) + E_{High}(\beta_T)) \tag{15}$$

are the required queries that correspond to $I_1(\beta_T)$ and $I_2(\beta_T)$ respectively, the total number of queries is

$$Q(\beta_T) = Q_1(\beta_T) + Q_2(\beta_T) \tag{16}$$

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

Evaluation. The results of the mathematical prediction and a simulation are depicted in Figure 2. We computed the expected number of queries for $\beta_T \in \{2, \ldots, 25\}$ for the parameter sets `kem/mceliece348864`, `kem/mceliece460896`, `kem/mceliece6688128`, `kem/mceliece6960119`, and `kem/mceliece8192128` of "Classic McEliece" (see Table 2). In addition, we implemented the described iterative chunking strategy using Python and SageMath. We ran this implementation as a simualtion with a ideal oracle which always returns the correct response. The simulation was applied to ten different key pairs using ten different plain-/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. The results of the simulation and the later experiments in Section 4.3 show that the estimate matches very well. The number of traces are decreased to approximately 12.3% for `kem/mceliece348864` down to 10% for `kem/mceliece8192128` compared to the ISD-supported iterative approach in Algorithm 2. As predicted, the number of traces depends on the choice of the chunk size threshold β_T. A smaller chunk size causes more queries. If the chunk size is too large, the bucket gets filled up with not inspected, as 'high' responded queries. Thus, additional traces have to be acquired to collect enough columns to apply a plain ISD. We could also spend more computing power to apply Stern or MMT as mentioned above to lower the number of traces further.

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 prediction are very close to each other.

3.4 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

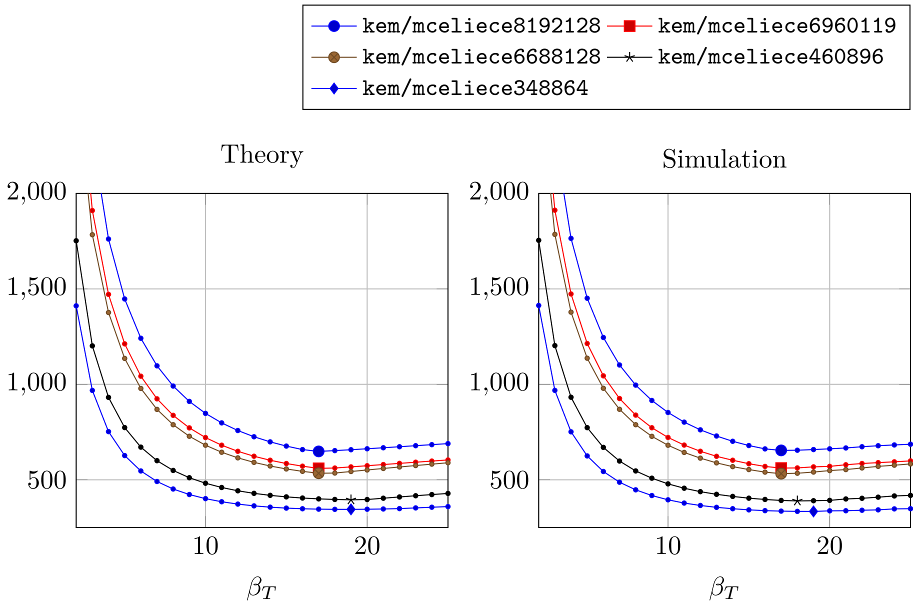

Fig. 2. The expected number of queries needed for threshold values from 2 to 25 both for the theoretical prediction and averaged simulations. The minima are marked.

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 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. A comparison to plain ISD can also be found in Table 5.

4 Attack Evaluation

For the practical attack evaluation we adapted the implementation presented in [37] 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.

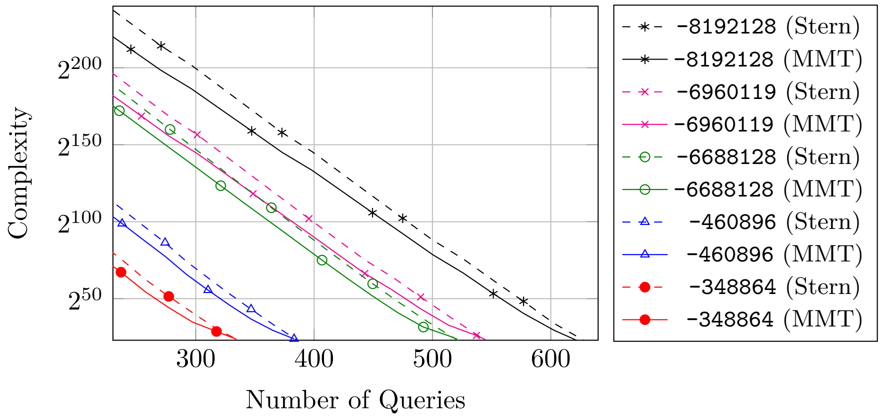

Fig. 3. Time-queries trade-off when using ISD decoding algorithms.

4.1 Leakage Analysis

To construct a decryption oracle for our attack approach we investigated the implementation by Wang et al. from [37] 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 [37] 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

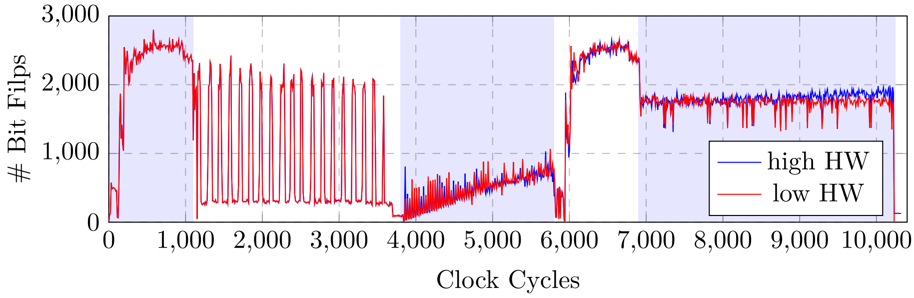

Fig. 4. Simulated power consumption based on a Hamming-distance model for successful decoding (high HW in the resulting error vector) and unsuccessful decoding (low HW) using a small parameter set with $m = 12$, $t = 66$, and $n = 3307$. The different parts of the algorithm can clearly be seen (marked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

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.

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 [38] to compare the means of the traces from the error-vector construction

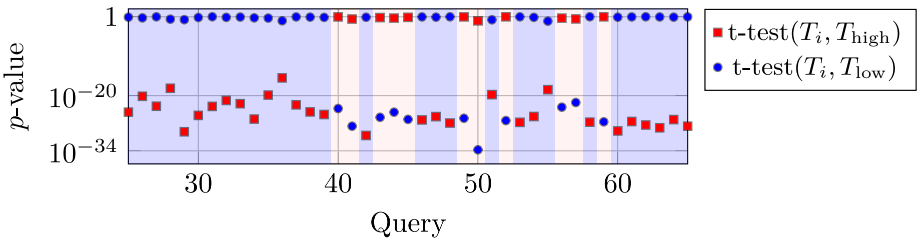

Fig. 5. Example p-values of the t-tests of consecutive oracle queries. For each query a trace T_i is compared to a known faulty (T_{low}) and a known decodable trace (T_{high}). If a trace is similar to the faulty trace (i.e. due to a higher p-value), a decoding failure is detected (light blue background).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

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}:

$$p_\Delta = \text{t-test}(T_i, T_{high}) - \text{t-test}(T_i, T_{low}) \tag{17}$$

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 decryption oracle therefore is:

$$response = \begin{cases} true \text{ (decodable)}, & \text{if } p_\Delta > 0 \\ false \text{ (not decodable)}, & \text{otherwise} \end{cases} \tag{18}$$

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

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}) ≈ 0;
return T'_{high}, T'_{low};

Algorithm 6: DEMA-based Oracle

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 $\begin{cases} true & if\ p_\Delta > 0 \\ false & otherwise \end{cases}$;

as described in [20]. 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 set clock frequency of 24 MHz 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 '1's 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 a trace that is recorded if a single '1' is recovered.

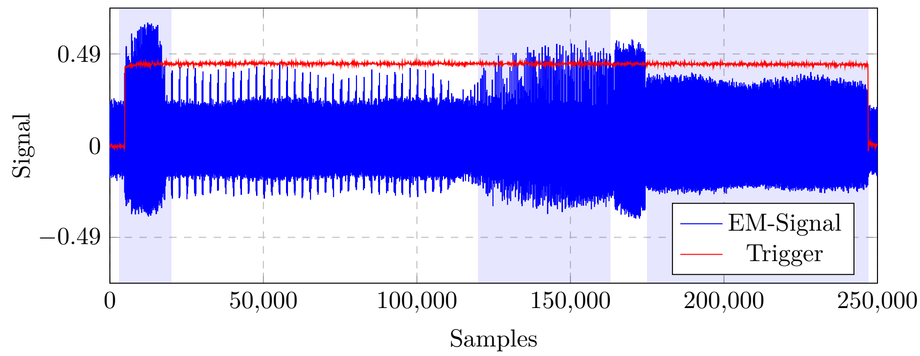

Fig. 6. Example EM trace of the decryption process on a SAKURA-X (Xilinx Kintex-7, XC7K160T). The five steps (additive FFT, double syndrome computation, Berlekamp-Massey, additive FFT, error vector construction) are highlighted with alternating blue background color as in Figure 4.

4.3 Practical Evaluation

To evaluate the attack in practice, we ported the FPGA design of [37] 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 $(g(x), (\alpha_0, \alpha_1, \dots \alpha_{n-1}))$, 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[4].

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). Figure 6 shows 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 3.2) 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 some computing power, he is able to reduce the number of traces even further, trading in a higher complexity of the ISD.

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

We are using the same implementation of the iterative chunking as the simulation described in Section 3.3 — however, we substituted the perfect 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 plain-/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 of the second round of the NIST competition, 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.

5 Conclusion

In this paper we presented a side-channel information-based plaintext recovery attack against the Niederreiter cryptosystem and specifically the "Classic McEliece" NIST submission. We introduced a novel reaction attack using an iterative chunking strategy and basic Information Set Decoding which enables the comulative recovery of error vector values within a single decryption oracle request. This approach decreases the required oracle requests to approximately 12.3% for `kem/mceliece348864` (335 traces) down to 10% for `kem/mceliece8192128` (654 traces) compared to the simple ISD-supported iterative approach which requires $k + 2$ requests.

We mathematically predict the optimal value for the parameter β_T which is used in the iterative chunking. We implemented the chunking strategy and exhaustively simulate the approach for multiple values using an ideal decryption oracle. The simulated results meet the predicted values very well. Additionally, we successfully demonstrated the feasibility of the attack for all proposed parameter sets of the second round using EM side-channel leakage from the reference FPGA implementation of the "Classic McEliece" submission which uses a Berlekamp-Massey decoder.

With our attack we showed that a side channel information on the decryption result can be efficiently exploited to recover a plaintext. 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.

`kem/` `mceliece-`	Approach	Simulation min.	avg.	max.	Theory plain	cost $\approx 2^{40}$	Experiment
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 plain-/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 querries.

A Appendix

A.1 Proof of Proposition 1

a. Since the probability of hitting a low chunk is $\Pr(Low) = \frac{\binom{n-w}{\beta}}{\binom{n}{\beta}}$, we obtain $p_\beta = 1 - \frac{\binom{n-w}{\beta}}{\binom{n}{\beta}}$.

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

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

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

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

A.2 Proof of Proposition 2

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) position of the chunk. Hence $E_{High}(\beta|w(\beta) = 1) = \log_2 \beta + 1 - 1/\beta$.

For $E_{High}(\beta|w(\beta)=2)$ we have:

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

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.

Acknowledgements

This work has been supported by the European Commission through the ERC Starting Grant 805031 (EPOQUE). Furthermore, this research 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.

References

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

[2] 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.

[3] 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.

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

[5] 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.

[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 2.* `https://classic.mceliece.org/nist/mceliece-20190331.pdf`. Mar. 2019.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

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

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

[28] 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.

[29] 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.

[30] P. W. Shor. "Algorithms for quantum computation: Discrete logarithms and factoring". In: *Foundations of Computer Science – FOCS 1994*. IEEE, 1994, pp. 124–134.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] W. Wang, J. Szefer, and R. Niederhagen. “FPGA-based Key Generator for the Niederreiter Cryptosystem Using Binary Goppa Codes”. In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by W. Fischer and N. Homma. Vol. 10529. LNCS. Springer, 2017, pp. 253–274.

[37] 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.

[38] 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.