QC-MDPC: A Timing Attack and a CCA2 KEM

Edward Eaton[1], Matthieu Lequesne[2][3], Alex Parent[1], and Nicolas Sendrier[3]**

[1] ISARA Corporation, Waterloo, Canada
{ted.eaton,alex.parent}@isara.com
[2] Sorbonne Universités, UPMC Univ Paris 06, France
[3] Inria, Paris, France
{matthieu.lequesne,nicolas.sendrier}@inria.fr

Abstract. In 2013, Misoczki, Tillich, Sendrier and Barreto proposed a variant of the McEliece cryptosystem based on quasi-cyclic moderate-density parity-check (QC-MDPC) codes. This proposal uses an iterative bit-flipping algorithm in its decryption procedure. Such algorithms fail with a small probability.
At Asiacrypt 2016, Guo, Johansson and Stankovski (GJS) exploited these failures to perform a key recovery attack. They introduced the notion of the *distance spectrum* of a sparse vector and showed that the knowledge of the spectrum is enough to find the vector. By observing many failing plaintexts they recovered the distance spectrum of the QC-MDPC secret key.
In this work, we explore the underlying causes of this attack, ways in which it can be improved, and how it can be mitigated.
We prove that correlations between the spectrum of the key and the spectrum of the error induce a bias on the distribution of the syndrome weight. Hence, the syndrome weight is the fundamental quantity from which secret information leaks. Assuming a side-channel allows the observation of the syndrome weight, we are able to perform a key-recovery attack, which has the advantage of exploiting all known plaintexts, not only those leading to a decryption failure. Based on this study, we derive a timing attack. It performs well on most decoding algorithms, even on the recent variants where the decryption failure rate is low, a case which is more challenging to the GJS attack. To our knowledge, this is the first timing attack on a QC-MDPC scheme.
Finally, we show how to construct a new KEM, called ParQ that can reduce the decryption failure rate to a level negligible in the security parameter, without altering the QC-MDPC parameters. This is done through repeated encryption. We formally prove the IND-CCA2 security of ParQ, in a model that considers decoding failures. This KEM offers smaller key sizes and is suitable for purposes where the public key is used statically.

Keywords: post-quantum cryptography, code-based cryptography, QC-MDPC codes, side-channel attack, timing attack, CCA2 security, key encapsulation

** Supported in part by the Commission of the European Communities through the Horizon 2020 program under project number 645622 (PQCRYPTO).

1 Introduction

Code-based cryptography is almost as mature as public-key cryptography itself, dating back to 1978 with the invention of the original McEliece public-key encryption scheme [28]. This scheme, when used with (as originally proposed) binary Goppa codes, has largely resisted all cryptanalytic efforts, from both classical and quantum adversaries. Because of this, code-based cryptography is a strong candidate for post-quantum standardisation, with several variants [31,30] attempting to make improvements or refinements on the original design.

Following [18,4], a new variant was proposed in 2013 using quasi-cyclic (QC) moderate density parity-check (MDPC) codes [29]. QC-MDPC codes use much shorter keys (about 10 kbits). This choice appears promising and the QC-MDPC scheme was recommended for further study by the report "Initial Recommendation of long-term secure post-quantum systems" of the European project PQCRYPTO [3]. Some hardware implementations of this scheme were published in [22] and [27].

The decryption algorithm of the QC-MDPC scheme is a variant of Gallager's bit-flipping algorithm [19]. It is an iterative algorithm with a simple structure, very easy to implement, even on constrained devices. It has an inconvenient though, it is subject to failure with non-negligible probability. The algorithm proposed in the original paper [29] has a *decoding failure rate* (DFR) of 10^{-7}.

While decoding errors may not represent a serious reliability issue, in a recent paper by Guo, Johansson and Stankovski (GJS) [20], the authors showed that these decoding failures actually do represent a very serious security issue. The authors exploited this DFR and managed to successfully recover the key by analyzing the error patterns that made the decryption fail. They found that these error patterns are correlated with the key. They introduce a new tool, the *distance spectrum*, to describe the correlation. They successfully use this correlation to perform their attack and give some hints on the reason why error patterns correlated in such a way are more prone to cause decryption failure.

The original QC-MDPC primitive is extremely vulnerable because the adversary may choose the error and even force a higher weight, in this case the attack of [20] recovers the key within minutes, when attacking a parameter set intended for the 80-bit classical security level. With a semantically secure conversion (CCA security, as in [23]) it requires $2^{39.7}$ operations.

1.1 Our Contributions

In this paper we extend the analysis of the GJS attack on QC-MDPC. The GJS attack works because the decoding failure depends of the existence of common values in the spectrums of the error pattern and of the secret key. In Section 3 we show that this correlation can be observed through the weight distribution of the first syndrome computed by the MDPC decoder. Pushing the analysis further we are able to quantify this bias. This allows us to perform a side-channel attack using the syndrome weight to recover the distance spectrum of the secret key. We show that the number of samples we need to make this attack work is consistent with the Chernoff bound applied to the above mentioned bias. This opens the way to theoretical estimates for the cost of attacks related to the secret key distance spectrum recovery.

Next, by remarking that the syndrome weight is correlated to the decoding time, we perform a GJS type of attack by counting the number of iterations. This provides a timing attack which is very generic and can be applied to any variant of the bit flipping algorithm which is not protected against timing attacks. Moreover, it works regardless of the failure rate. To our knowledge, this is the first timing attack on this kind of scheme. This confirms a conjecture made by Maurich and Güneysu [25] that the number of iterations in the decoding procedure leaks secret information.

In Section 4 we demonstrate the power of this attack by showing experimental results of the timing attack on various parameter sets and decoding procedures. This shows that the attack is practical even against the 256-bit classical parameter set. Additionally, we analyze and discuss how some other variations in the decoding procedure proposed in [27] affect the attack and its effectiveness.

Finally in Section 5, we show a new construction for a QC-MDPC-based KEM, called ParQ. This KEM uses QC-MDPC encryption as the underlying primitive, and does not need to alter the parameter set of the primitive itself. The scheme works by creating multiple independent encapsulations of the same key, so that a decapsulation failure only occurs if a decoding failure happens for each ciphertext. This causes the decapsulation algorithm to only fail with negligible probability, and so it entirely eliminates the possibility of using decoding failures to recover the key with the GJS attack. This scheme does not increase key sizes at all, and only increases the size of the encapsulation by a small factor $(3-12\times)$. We provide a comprehensive proof of IND-CCA2 security of the scheme, and analyse the KEM compared with other code-based key transport methods. Our proof considers the possibility of decoding failures. Other CCA2 constructions [23,26] did not consider this, which is why the GJS attack was able to break CCA2 security. Most commentary on mitigating the GJS attack has focused on either altering the parameters of QC-MDPC to decrease the DFR or using the keys ephemerally. Through our scheme we show that there is a third option that can address decoding failures at the protocol level.

1.2 Related Work

The McEliece cryptosystem was originally proposed in [28], and low density parity-check codes were proposed in [19]. The QC-MDPC variant of McEliece was proposed in [29]. The key-recovery reaction attack we focus on in this paper was shown in [20]. In [17], the authors analyzed how the observations from [20] applied to the case of LDPC McEliece [30], showing that the attack also worked on soft decision decoding procedures.

Since the first publication of the QC-MDPC scheme, efforts have been made to tune the decoding algorithm, especially exploring the different ways to fix the thresholds in order to reduce the DFR [10]. This is discussed in details in Section 2.2.

Side-channel timing attacks [24] on McEliece systems other than QC-MDPC have been considered for example in [33,34,35] which has motivated the need for constant-time implementations [7,13]. In [25,11,12], the authors demonstrated several power-analysis side-channel attacks on QC-MDPC, and [25] conjectured that it might be possible that the number of decoding rounds leaks secret information. To our knowledge, our paper is the first to conclusively show that this is in fact the case.

CCA2 conversions for McEliece systems have been considered before, most notably in [23]. General conversions for designing CCA2 KEMs from OW-CPA systems were studied in [15]. Other key exchange and key encapsulation schemes related to QC-MDPC include [5,13,14,26].

A line of constructions beginning with [32], and applied to McEliece in several follow-up works [16,36] explored the concept of the k-repetition paradigm for encryption. This paradigm bears some resemblance to our parallel KEM in Section 5, although these constructions are different and have a different goal: CCA2 security without random oracles.

2 QC-MDPC McEliece and the GJS Attack

2.1 Quasicyclic Moderate Density Parity Check McEliece

QC-MDPC-McEliece is a public key encryption method consisting of three algorithms. It is defined by four parameters, n, k, w, and t. The key generation algorithm $\mathsf{QCMDPC.KeyGen}$ constructs an (n, k)-linear quasicyclic code, consisting of a generator matrix G (the public key) and a parity check matrix H (the secret key), for which each row has weight w. The encryption algorithm $\mathsf{QCMDPC.Enc}$ encrypts a plaintext $x \in \mathbb{F}_2^k$ by calculating the corresponding codeword to x, xG and adding an error e of weight t to obtain the ciphertext $c = e + xG$. The decryption algorithm $\mathsf{QCMDPC.Dec}$ decodes c back to xG and recovers x.

While QC-MDPC can allow for k to be any divisor of n, we will consider the case of $n/k = 2$. We let $\mathbb{E}$ denote the set of $e \in \mathbb{F}_2^n$ with Hamming weight t. Note that the size of each block, $r = (n - k) = k$.

Algorithm 1 $\mathsf{QCMDPC.KeyGen}$

Input: Security parameter 1^λ.
Output: Public key pk, secret key sk.

```
1: Generate $h_0, h_1 \in \mathbb{F}_2^k$, both with weight $w/2$.
2: Let $H = [H_0|H_1]$, where $H_0$ and $H_1$ are $k \times k$ matrices generated from $h_0$ and $h_1$
   by cyclically rotating them.
3: Set $G = [I_k|Q]$, where $I_k$ is the $k \times k$ identity matrix, $Q = (H_1^{-1}H_0)^T$.
4: **return** $pk = q$, the first row of $Q$ and $sk = h_0, h_1$. These allow for the reconstruction of $G$ and $H$.
```

Algorithm 2 QCMDPC.Enc

```
Input: Public key $pk = q$, plaintext $x \in \mathbb{F}_2^k$, error vector $e \in \mathbb{E}$.
Output: Ciphertext $c \in \mathbb{F}_2^n$.
1: Reconstruct $G = [I_k|Q]$ by cyclically rotating $q$ to obtain $Q$.
2: return $c = e + xG$.
```

Algorithm 3 QCMDPC.Dec

```
Input: Secret key $sk$, public key $pk$, and ciphertext $c \in \mathbb{F}_2^n$.
Output: Plaintext $x \in \mathbb{F}_2^k$ and error vector $e \in \mathbb{E}$, or decryption failure symbol $\perp$.
1: Reconstruct parity-check matrix $H = [H_0|H_1]$, and generator matrix $G = [I_k|Q]$.
2: Run the decoding procedure on $c$ with parity-check matrix $H$ to recover codeword
   $xG$. If a decoding failure occurs, return $\perp$.
3: Recover $x$ from the first $k$ bits of $xG$.
4: Recover $e = c - xG$.
5: return $(x, e)$.
```

Multiple parameter sets for QC-MDPC have been proposed for multiple security levels. Our interest is in the 80-bit and 256-bit classical security sets (which corresponds to at least 40-bit and 128-bit quantum security) that were originally proposed in [29], and have been further discussed in [5].

Classical bit-strength	n	k	w	t
80	9602	4801	90	84
128	20 326	10 163	142	134
256	65 542	32 771	274	264

2.2 QC-MDPC Decoding Procedure

The original paper on MDPC codes [29] proposes to use a hard decision version of Gallager's bit-flipping algorithm for decoding LDPC codes [19]. The main idea is the following. At each iteration, the algorithm computes the number of unsatisfied parity-check equations associated to each bit. Each bit that is involved in $\geq b$ unsatisfied equations is flipped, for b some threshold, and the syndrome is recomputed. This repeats until the syndrome becomes zero. In practice, the algorithm stops after fixed number of iterations and this is considered a decoding failure.

For our main analyses we use decoder $\mathcal{D}_1$ from [27] with fixed thresholds $\{95, 85, 80, 76, 74, 73, 72, 72\}$. $\mathcal{D}_1$ is a modification of Gallager's algorithm which updates the syndrome in place after each bit flipped. Algorithm 4 is the normal out-of-place bit flipping algorithm and Algorithm 5 is the in-place version.

Variable thresholds. A more recent approach, studied in [10], is to choose the values of b at each iteration depending on the syndrome weight at the time. This

Algorithm 4 Iterative bit flipping decoding algorithm

```
Input: $c = (c_0, \dots, c_{n-1}) \in \mathbb{F}_2^n$, $H = (h^{(0)}, \dots, h^{(n-1)}) \in \mathbb{F}_2^{r \times n}$
  $s \leftarrow H \cdot c^\intercal$                                   ▷ compute the syndrome
  while $s \neq 0$ do
    for $i = 0, \dots, n-1$ do
      if $\langle s, h^{(i)} \rangle \geq b$ then      ▷ if number of unsatisfied equations $\geq$ threshold $b$
        $c_i \leftarrow c_i \oplus 1$                                  ▷ flip the $i^{\text{th}}$ bit
    $s \leftarrow H \cdot c^\intercal$
  return $c$
```

Algorithm 5 In-Place: Iterative bit flipping decoding algorithm

```
Input: $c = (c_0, \dots, c_{n-1}) \in \mathbb{F}_2^n$, $H = (h^{(0)}, \dots, h^{(n-1)}) \in \mathbb{F}_2^{r \times n}$
  $s \leftarrow H \cdot c^\intercal$                                   ▷ compute the syndrome
  while $s \neq 0$ do
    for $i = 0, \dots, n-1$ do
      if $\langle s, h^{(i)} \rangle \geq b$ then      ▷ if number of unsatisfied equations $\geq$ threshold $b$
        $c_i \leftarrow c_i \oplus 1$                                  ▷ flip the $i^{\text{th}}$ bit
        $s \leftarrow s \oplus h^{(i)}$
  return $c$
```

approach gives the best results so far, both in terms of decryption failure rate and average number of iterations.

In both cases, until now the thresholds were claimed as experimental results with no explanation on the way they were generated. In appendix B we discuss a procedure to obtain such thresholds for any security parameters.

2.3 The GJS Attack

The key recovery attack in [20] is a reaction attack. It takes advantage of the decoding failures that occasionally occur during decryption. It assumes only that an adversary is able to tell when such an error has occurred, for example because a request for resend is sent back. It consists of two steps. The first step is to calculate the *distance spectrum* of the secret key (or one part of the secret key), based on observing a large number of error vectors that resulted in a decoding failure. The second step is to reconstruct the secret key based on its distance spectrum.

In this paper, we will focus our attention on the first step. Reconstructing the secret key from the distance spectrum has been analysed before [20,17], and shown to be fairly fast and simple as compared to the first step, and is an entirely offline computation, requiring no communication.

Definition 1 (Distance Spectrum). *The distance spectrum of a vector $h \in \mathbb{F}_2^r$, denoted $\Delta(h)$, is the set of distances δ such that there exist two non-zero* *bits of h at distance δ. The distance are counted cyclically.*

$$\Delta(h) = \left\{ \delta : 1 \leq \delta \leq \left\lfloor \frac{r}{2} \right\rfloor, \exists(i,j), \begin{array}{l} 0 \leq i < j < r, \\ h[i] = h[j] = 1, \\ \min\{j-i, r-(j-i)\} = \delta \end{array} \right\}$$

where $h[i]$ denotes the i^{th} entry of the binary vector h.

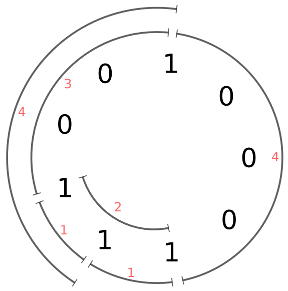


Fig. 1: Distance spectrum of 100100011_2

For example, the distance spectrum of the vector 100100011_2 is $\{1,2,3,4\}$ (fig. 1). Note that any cyclic shift or reversal of a vector will result in the same distance spectrum. In [20,17], it was shown how to quickly reconstruct a vector (up to a reversal or cyclic shift) from a distance spectrum. The first step of the GJS attack is to find the distance spectrum of the first half h_0 of the secret key (h_0, h_1). From this, h_0 can be computed, which allows us to also calculate h_1 by elementary linear algebra.

In order to analyse more precisely the results, we need to take into account the fact that some distances may appear more than once.

Definition 2 (Distance Spectrum with multiplicity). *The distance spectrum with multiplicity of a vector $h \in \mathbb{F}_2^r$, denoted $\Delta^\mu(h)$, is a vector of $\mathbb{N}^{\lfloor \frac{r}{2} \rfloor}$ such that for every distance $1 \leq \delta \leq \left\lfloor \frac{r}{2} \right\rfloor$, its δ^{th} component $\Delta^\mu(h)[\delta]$ is the number of existing sets of two non-zero bits of h at distance δ. The distance are counted cyclically.*

Example 1. For $h = 0011000011_2$ (see Figure 1), then $\Delta^\mu(h) = [2,1,1,2]$.

In general we can see that if a vector $h_0 \in \mathbb{F}_2^k$ has weight w_0, then the distance spectrum with multiplicity of h_0 will be a vector of size $\lfloor k/2 \rfloor$ such that the sum of the entries of $\Delta^\mu(h_0)$ is $\binom{w_0}{2}$.

Finding the distance spectrum of the secret key is done by taking note whether a decoding failure occurs for a large number of error vectors. This is done because of the following observation:

Observation 1 (GJS, Key Observation). When a distance in the error vector used in a QC-MDPC encryption matches a distance in the distance spectrum of the secret key, a decoding failure is *less* likely to occur.

Based on this observation, it was noticed that by carefully calculating the decoding failure rate for errors that have a given distance vs. those that do not, the multiplicity of that distance in the secret key's distance spectrum can be correctly guessed. Note that this observation applies to each half of the error vector (and parity check matrix) independently. So when we refer to the distance spectrum of the error or parity-check matrix, we mean the distance spectrum of the first k bits, unless stated otherwise.

Algorithm 6 was proposed in [20] for attacking the CCA security of a QC-MDPC implementation.

Algorithm 6 GJS CCA attack

```
1: Initialize $observed_d = 0$ and $failed_d = 0$ for $d \in \{1, \dots, \lfloor k/2 \rfloor\}$.
2: for $i = 1$ to $M$ do
3:     Send $c = \mathsf{QCMDPC.Enc}(x, e)$ with a uniformly random $e = [e_0 || e_1]$ to target.
4:     for $d \in \Delta(e_0)$ do
5:         Increment $observed_d$ by 1.
6:         if Decoding failed for $c$ then
7:             Increment $failed_d$ by 1.
8: return $failed_d / observed_d$ for $d \in \{1, \dots, \lfloor k/2 \rfloor\}$.
```

The resulting values, $failed_d/observed_d$ for each d give an estimate of the decoding failure rate for error vectors with d in their distance spectrum. We can then recover the distance spectrum, identifying the multiplicity of each distance from the following observation:

Observation 2 (GJS). For a fixed key, the decoding failure rate for error vectors with d in their distance spectrum is inversely proportional to the multiplicity of d in the distance spectrum of the key.

For large enough values of M, the decoding failure rate clearly separates into bands. These bands exactly correspond to the multiplicity of that distance in $\Delta(h_0)$. This allows an attacker to recover $\Delta(h_0)$, and thus the secret key.

The complexity of the attack is dominated by the value M. The decoding failure rates for different multiplicities are quite close together, and so a very accurate estimation is need in order to properly decide on the multiplicity. In [20], the authors found that $M = 2^{29}$ was sufficient for the 80-bit classical parameter set, using the Gallager decoding algorithm. They conjectured that using a more sophisticated decoding algorithm like that in [29], would mean that M would have to be increased by an amount proportional to the difference in the decoding failure rate. They also conjectured that higher parameter sets would not significantly alter the effectiveness of the attack, as the decoding failure rate does not significantly change.

3 Analysis and Timing Attack

3.1 Correlation

Our attack is based on the fact that the average syndrome weight is slightly different if the relative position of non-zero bits in the key and the error are correlated.

For the sake of simplicity, in this section, we will consider a parity-check matrix made of one single circulant block in $H \in \mathbb{F}_2^{k \times k}$ instead of two. We will see later that the practical results are the same. We denote by $h \in \mathbb{F}_2^k$ the first row of the matrix H. The variable t still represents the weight of the error e, so here the numerical value of t should be half its usual value.

Without any information. Let us suppose that we do not have any information on the key. For a random key vector h of size k and weight d and a random error vector e of size k and weight t, denote by $f(k,d,t,b)$ the probability that the scalar product in $\mathbb{F}_2$ is of parity b:

$$f(k,d,t,b) := \Pr[\langle h, e\rangle = b] = \sum_{i=0,\ i \text{ is of parity } b}^{d} \frac{\binom{d}{i}\binom{k-d}{t-i}}{\binom{k}{t}}.$$

The average syndrome weight of an error e and a parity-check matrix generated by cyclic shifts of h is k times the probability that a bit is non-zero (see [9, page 91]), that is:

$$\mathbb{E}\left[\,\mathsf{wt}\left(H \cdot e^{\intercal}\right)\right] = k \cdot f(k,d,t,1).$$

Case of two consecutive non-zero bits in the key. Now, suppose the key vector h has ℓ times two consecutive non-zero bits. Let us observe the shifts of the vector:

$$\begin{array}{lll}
\mathsf{shift}(h) = \boxed{1}\boxed{1}\boxed{\qquad u, \mathsf{wt}(u) = d-2 \qquad} & & \ell \text{ times} \\
\mathsf{shift}(h) = \boxed{1}\boxed{0}\boxed{\qquad u, \mathsf{wt}(u) = d-1 \qquad} & & d-\ell \text{ times} \\
\mathsf{shift}(h) = \boxed{0}\boxed{1}\boxed{\qquad u, \mathsf{wt}(u) = d-1 \qquad} & & d-\ell \text{ times} \\
\mathsf{shift}(h) = \boxed{0}\boxed{0}\boxed{\qquad\ \ u, \mathsf{wt}(u) = d \qquad\ \ } & & k-2d+\ell \text{ times.}
\end{array}$$

Suppose that the first two bits of the error vector are non-zero, that is:

$$e = \boxed{1}\boxed{1}\boxed{\qquad u, \mathsf{wt}(u) = t-2 \qquad}\,.$$

With this extra assumption on the form of h and e, the average syndrome weight of e with respect to the the parity-check matrix H generated by cyclic shifts of h can now be approximated by:

$$\begin{aligned}\mathbb{E}\left[\,\mathsf{wt}\left(H\cdot e^{\mathsf{T}}\right)\right] = \quad & \ell \quad f(k-2,d-2,t-2,1)\\ + \quad & 2(d-\ell) \quad f(k-2,d-1,t-2,0)\\ + \; & (k-2d+\ell)\; f(k-2,d,t-2,1).\end{aligned} \tag{1}$$

Contrary to the previous result, this is an approximation. Indeed, this model assumes that the rest of the vector (denoted by u) is random for each shift. It does not take into account the covariance between the bits of the syndrome. Previously we were averaging on all the lines and the covariance was therefore null, while here the fact that we group the rows depending on the value of the first two bits breaks the symmetry. Still, we will see that the approximation is close to the real value and we can neglect the correction term for the rest of the study.

Exploiting the leak. Suppose that we only consider error patterns starting with two consecutive non-zero bits, the syndrome weight is expected to be slightly different on average, depending on ℓ the number of times two consecutive bits are non-zero in the key vector h. Moreover, the expected value varies linearly with ℓ. Therefore, if we observe enough values of the syndrome weight, we can recover the value of ℓ.

Definition 3 (Average syndrome weight with multiplicity). *Let us denote by D_ℓ the following set:*

$$D_\ell := \left\{(h,e) \in \mathbb{F}_2^k \times \mathbb{F}_2^k \,|\, \mathsf{wt}(h) = d, \mathsf{wt}(e) = t, \delta \in \Delta(e), \Delta^\mu(h)[\delta] = \ell\right\}.$$

The average syndrome weight with multiplicity $\bar{\sigma}_\ell$ is the expectation of the syndrome weight for a uniform distribution of (h,e) over D_ℓ:

$$\bar{\sigma}_\ell := \mathbb{E}_{(h,e)\sim\mathcal{U}(D_\ell)}\left[\,\mathsf{wt}(H\cdot e^{\mathsf{T}})\,\right].$$

From the equation (1) in Section 3.1 we know that we can approximate $\bar{\sigma}_\ell$ by:

$$\begin{aligned}\bar{\sigma}_\ell = \quad & \ell \quad f(k-2,d-2,t-2,1)\\ + \quad & 2(d-\ell) \quad f(k-2,d-1,t-2,0)\\ + \; & (k-2d+\ell)\; f(k-2,d,t-2,1).\end{aligned}$$

with $f(k,d,t,b) := \displaystyle\sum_{i=0,\ i \text{ is of parity } b}^{d} \frac{\binom{d}{i}\binom{k-d}{t-i}}{\binom{k}{t}}$

Comparison with measured values. The values of $\bar{\sigma}_\ell$ correspond to the different clusters that we can see on the figures. According to the approximation, the value of $\bar{\sigma}_\ell$ is linear in the multiplicity: $\bar{\sigma}_0 - \bar{\sigma}_\ell = \ell\cdot(\bar{\sigma}_0 - \bar{\sigma}_1)$. This is consistent with what we observe..

With the usual parameters for 80-bit security, (here using $t = 42$ as there is only one block) we obtain $\bar{\sigma}_0 = 1324.23$ and $\bar{\sigma}_1 = 1323.28$.

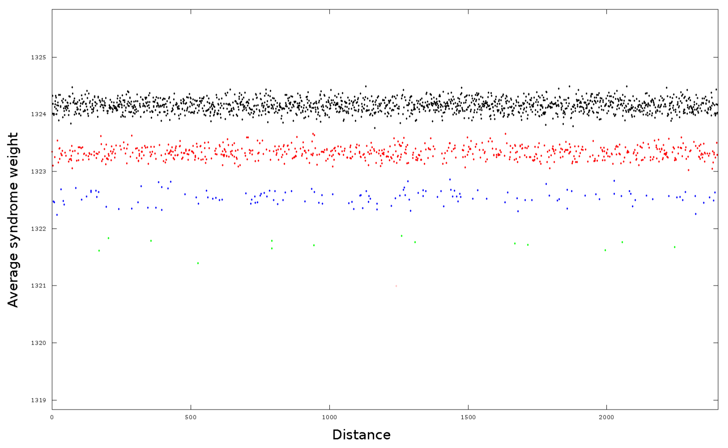


Fig. 2: Attack on the syndrome weight (1 block): Average syndrome weight per distance, 10^5 samples. The color of the distances indicate their multiplicity in the key spectrum (black = 0, red = 1, blue = 2, green = 3)

When comparing the values to those measured on Fig. 2, we can see that the measured $\bar{\sigma}_0$ is slightly lower than the approximated value, and on the contrary $\bar{\sigma}_1$ is slightly higher. This error is due to the approximation that neglects the covariance. When performing the same experiment on parameters for LDPC codes, where the covariance is much smaller, the measures correspond exactly to the computed values.

As a consequence, the real distance $\bar{\sigma}_0 - \bar{\sigma}_1$ is smaller than the one computed using equation (1). Hence, the theoretical analysis gives an interesting bound on the relative distance $\varepsilon = \frac{\bar{\sigma}_0 - \bar{\sigma}_1}{k}$: $\varepsilon_{\text{measured}} < \varepsilon_{\text{computed}}$.

Hypothesis testing. Each syndrome is the result of k scalar products between the error and a parity-check equation. When the error contains a distance present in the spectrum of the key with multiplicity ℓ, the average syndrome weight is $\bar{\sigma}_\ell$, this means that on average $\bar{\sigma}_\ell$ of the k parity-check equations are not verified. Hence, under the independence assumption, we can see each bit of the syndrome as a Bernoulli trial satisfied with probability $\frac{\bar{\sigma}_\ell}{k}$.

Here, our goal is to decide for each distance δ whether or not δ is in the distance spectrum of h. We do not care about the multiplicity. Formally, we want to distinguish D_0 from $\cup_{\ell \geq 1} D_\ell$. Let us by $D_{\geq 1} := \cup_{\ell \geq 1} D_\ell$. We can define $\bar{\sigma}_{\geq 1}$ on $D_{\geq 1}$ just like we defined $\bar{\sigma}_\ell$ on D_ℓ. The sets are disjoint so we have $\bar{\sigma}_{\geq 1} = \frac{\sum_{\ell \geq 1} \bar{\sigma}_\ell |D_\ell|}{\sum_{\ell \geq 1} |D_\ell|}$.

Hence, deciding whether a distance is in the spectrum of the key or not is just like distinguishing a random binary variable with success probability $p_0 := \bar{\sigma}_0$

from a random binary variable with success probability $p_1 := \bar{\sigma}_{\geq 1}$. This is a classic problem of hypothesis testing.

Note that for our parameters, the size of D_ℓ for $\ell \geq 2$ is negligible compared to D_1, hence there is no practical need to distinguish $\bar{\sigma}_1$ from $\bar{\sigma}_{\geq 1}$.

Sample size. There is a lot of literature about hypothesis testing, and in particular a theorem from Chernoff [21] concerning such cases.

Proposition 1 (Chernoff's bound). *Let $0 < p < 1$, let $X_1, X_2, \dots, X_N$ be independent binary random variables, with $\Pr[X_k = 1] = p$ and let $S_N = \frac{\sum_{k=1}^{N} X_k}{N}$. Then for any $t \geq 0$,*

$$\Pr[|S_N - p| \geq t] \leq 2\mathrm{e}^{-2Nt^2}.$$

This can be used to understand how the number of samples required to find the key evolves. Here we want to distinguish p_0 from p_1, we will use $\frac{p_0+p_1}{2}$ as the decision threshold. Chernoff's bound states that we should have $N \sim \frac{1}{\varepsilon^2}$ repeated Bernoulli trials for the decision test to be relevant, where $\varepsilon = |p_1 - p_0| = \frac{\bar{\sigma}_0 - \bar{\sigma}_1}{k}$ is the distance between the two outcomes.

To decide whether a particular distance δ is in the spectrum or not, we need to compute the mean of N Bernoulli trials, but each syndrome weight is already the sum of the results of k Bernoulli tests. Hence, we need to observe the weight of $\frac{N}{k}$ syndromes. These syndromes need to be in one of the D_ℓ, this means that the distance δ needs to be in the spectrum of the error pattern that generates the syndrome. As the error patterns are generated uniformly, we proceed by rejection sampling to ensure this condition. The number of vectors of size k and weight w that do not contain a particular distance is $\prod_{j=0}^{w-1}(k-3j)$, so neglecting the cases of multiplicity we obtain a good approximation of the frequency of such vectors with:

$$\alpha := \Pr(\delta \in \Delta(e)) \approx 1 - \frac{\prod_{j=0}^{\lfloor \frac{t}{2} \rfloor - 1}(k - 3j)}{\prod_{j=0}^{\lfloor \frac{t}{2} \rfloor - 1}(k - j)}.$$

Hence, to decide whether or not $\delta \in \Delta(h)$, we need to observe the decoding of $\frac{N}{\alpha \cdot k}$ syndromes, with $N \sim \frac{1}{\varepsilon^2}$. As we use the same data to decide for all distances, this is the number of samples needed to recover the whole spectrum.

3.2 Attack on the Syndrome Weight

Attack Model. The scenario for our attack is the following. Eve can encrypt random messages using the QC-MDPC scheme described in 2.1 and Alice's public key. She has access to the plaintext but cannot choose the messages. She sends the messages for decryption. Whenever the device decodes a message sent by Eve, she has a way to observe the weight of the syndrome.

The attack we describe here is an abstraction. We do not focus on how, or even if, Eve gets access to the data. It might be possible or not depending on a particular implementation and on the abilities of the attacker. The point is to establish through a simulation that some secret information leaks from the syndrome weight and to compare the cost of that simulation with the theoretical analysis of the previous section.

$$\begin{array}{lcr}
\textbf{Eve} & & \textbf{Alice's Decoder} \\
\hline
m \leftarrow \mathbb{F}_2^k & & \\
e \xleftarrow{\$} \mathbb{F}_2^n, \mathsf{wt}(e) = t & \xrightarrow{\quad c = G_{Alice} \cdot m^{\intercal} + e \quad} & \underline{\text{Decode}(c, H_{Alice}):} \\
& & s \leftarrow H_{Alice} \cdot c^{\intercal} \\
& \xleftarrow{\qquad\qquad \sigma \qquad\qquad} & \sigma \leftarrow \mathsf{wt}(s) \\
& & \dots
\end{array}$$

We suppose that Eve's error patterns are randomly generated. Indeed, in the scheme, semantically secure conversions ensure that the error patterns are random [23]. If we allow Eve to choose the error patterns, this will only make the attack easier, as in [20].

Contrary to [20], we collect information from all the error patters, not only those leading to a decoding failure.

Attack on Syndrome Weight. Our goal is to compute the distance spectrum of Alice's private key. For each distance δ between 1 and $\lfloor \frac{k}{2} \rfloor$ we want to decide whether or not $\delta \in \Delta(h_{Alice})$. As we have seen in 3.1, for each distance $\delta \in \Delta(e)$, the expected average weight of the syndrome $\sigma = \mathsf{wt}(s)$, where $s = H_{Alice} \cdot c^{\intercal} = H_{Alice} \cdot e^{\intercal}$, is expected to be different if $\delta \in \Delta(h_{Alice})$.

Hence, the idea is, for each distance δ, to compute the average value of the syndrome weight σ for error patterns e such that $\delta \in \Delta(e)$. The error patterns are generated randomly and each error e can be used to obtain information on all the distances in its spectrum. This leads to algorithm 7.

Following the discussion in Section 3.1, we will take $\mathsf{threshold} = \frac{\bar{\sigma}_0 + \bar{\sigma}_1}{2}$.

3.3 Attack on Iteration Count

Now that we know that the syndrome weight leaks information, any parameter correlated to this quantity could be used for a side channel attack. An interesting parameter that is often easy to measure is the number of iterations of a loop.

The decoding algorithm for QC-MDPC codes is an iterative algorithm with no termination proof. The number of rounds needed to correct the errors varies. This has been studied by in [10]. As mentioned in Section 2.2, the algorithm depends on the way we chose the thresholds. For most instances, using fixed or variable thresholds, the algorithm usually corrects the error in 3 rounds, but some instances need 4, 5 or even more iterations. Usual implementations abort after a certain number of rounds (around 10), this is what was used for the attack in [20].

Experimentally, we observe that the correlations between the spectrum of the error and the spectrum of the key has an impact on the average decryption time.

Algorithm 7 Computing the distance spectrum

```
Input: $N$ the size of the sample, oracle access to the decoder
  SyndromeCount $\leftarrow (0,\dots,0) \in \mathbb{N}^{\lfloor \frac{k}{2} \rfloor}$
  OccurenceCount $\leftarrow (0,\dots,0) \in \mathbb{N}^{\lfloor \frac{k}{2} \rfloor}$
  $\Delta \leftarrow (0,\dots,0)$
  for $0 \leq i \leq N-1$ do
      $e \xleftarrow{\$} \mathbb{F}_2^n, \mathsf{wt}(e) = t$
      $\sigma \leftarrow$ OracleDecoder$(e)$                          ▷ $\sigma = \mathsf{wt}(e \cdot H_{Alice}{}^\intercal)$
      for $\delta \in \Delta(e)$ do
          SyndromeCount$[\delta]$ += $\sigma$
          OccurenceCount$[\delta]$ += $1$
  for $1 \leq \delta \leq \lfloor \frac{k}{2} \rfloor$ do
      if SyndromeCount$[\delta]$/OccurenceCount$[\delta]$ < threshold then
          $\Delta[\delta] \leftarrow 1$
  return $\Delta$
```

The more distances appear both in spectrum of the error and in the spectrum and the key, the fewer the number of iterations needed to decode on average. This appears clearly on Fig. 3. We note that the correlation is slightly more important on Fig. 3 when we use variable thresholds than with fixed theresholds (the average value is lower for variable thresholds, but the same scale is used for both figures).

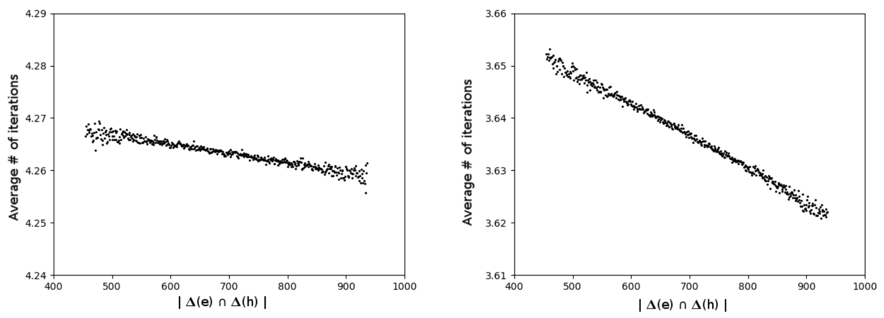


Fig. 3: Average number of iterations needed for decryption, depending on the size of the intersection of the spectrum of the error and the spectrum of the key. 2^{29} samples, 128-bit security QC-MDPC scheme, decoding with fixed thresholds (left) and variable thresholds (right). Note that use of variable thresholds results in stronger correlation.

This motivated us to try to perform a theoretical timing attack (algorithm 8). The scenario is the same as previously, but instead of observing the syndrome weight, Eve can measure the number of iterations needed to decode her message.

To obtain the spectrum, Eve uses the exact same data collection algorithm: for every distance in the spectrum, she computes the average number of iterations needed to correct an error containing this distance.

This works well and it is possible to fully recover the distance spectrum with variable thresholds using 2^{25} samples on 80-bit security QC-MDPC scheme, 2^{25} samples for 128-bit security parameters (see Fig. 6) and 2^{28} samples for 256-bit security parameters. For fixed thresholds, we manage to recover the spectrum for 256-bit security with 2^{28} samples.

Algorithm 8 Timing attack on QC-MDPC

```
 1: Initialize $observed_d = 0$ and $iterations_d = 0$ for $d \in \{1, \dots, \lfloor k/2 \rfloor\}$.
 2: for $i = 1$ to $M$ do
 3:     $e \xleftarrow{\$} \mathbb{F}_2^n, \mathsf{wt}(e) = t$
 4:     $c \leftarrow \mathsf{QCMDPC.Enc}(x, e)$
 5:     Send $c$ to target.
 6:     $n \leftarrow$ number of iterations (from side channel).
 7:     for $d \in \Delta(e_0)$ do
 8:         $observed_d$ += 1.
 9:         $iterations_d$ += $n$.
10: Return $iterations_d / observed_d$ for $d \in \{1, \dots, \lfloor k/2 \rfloor\}$.
```

4 Experimental Results

Results of Syndrome attack. The spectrum recovery algorithm was first tried on a simplified version of the scheme using only one block, in order to compare to the expected behaviour. The result is striking. Using the usual parameters for 80-bit security, with one hundred thousand samples, the spectrum appears very clearly and we can even see the multiplicities, that is, distances that appear several times in the key, see Fig. 2. When pushing to one billion samples, there is no room for confusion.

When performing the same experiment on the real QC-MDPC scheme with two blocks, we obtain similar results. The attack is performed on each block separately, that is for each error pattern, we added the syndrome weight to the counters of all distances present in the first half of the error to recover the spectrum of the first block. Because there is no correlation between the two halves of the error pattern, the presence of the second block acts as a random noise added to the syndrome weight. Hence the only difference is that we need more samples to reduce the variance and distinguish well which distances are in the key spectrum. Note that it is possible to compute the spectrum of both blocks at the same time, so there is no need to double the number of samples to recover the second block.

For 80-bit security parameters, we can see on Fig. 4 the spectrum appearing more and more distinctively when we increase the number of samples. With 2^{20}

samples, we can fully distinguish the spectrum. The same attack requires 2^{23} samples for 128-bit security parameters and 2^{25} for 256-bit security parameters.

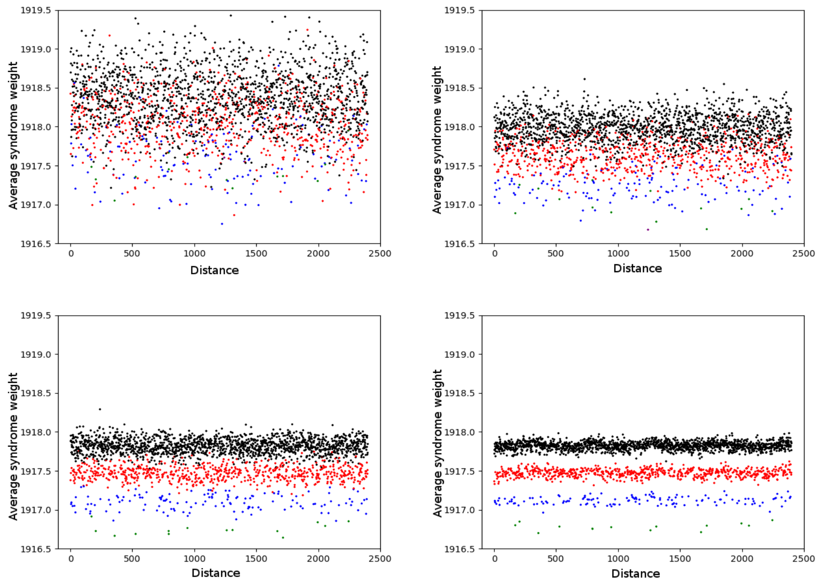


Fig. 4: Average syndrome weight per distance, (from left to right, from top to bottom) 2^{14}, 2^{16}, 2^{18} and 2^{20} samples, 80-bit security QC-MDPC scheme. The color of the distances indicate their multiplicity in the key spectrum (black = 0, red = 1, blue = 2, green = 3, purple $\geq$ 4)

This attack was also performed when another error is added to the syndrome, like in the Ouroboros scheme [14] (with an additional error of weight $3d$). Again, this only adds random noise and we can recover the spectrum with around a few million samples for the 80-bit security parameters.

Results of iteration attack. After running algorithm 8 we collect data corresponding to the average number of iterations it took to decode an error when d is present. The resulting plots (fig. 5) look very similar to the plots of the decoding failure rate that result from Algorithm 6. Once the bands have completely separated, the distance spectrum (and thus the secret key) can be recovered in the same way it was in the GJS [20] attack.

This side-channel attack is much faster than the reaction attack. An intuitive explanation for the speedup is that differences in the number of iterations are much more common than decoding errors. This allows more information about the correlations to be collected per iteration.

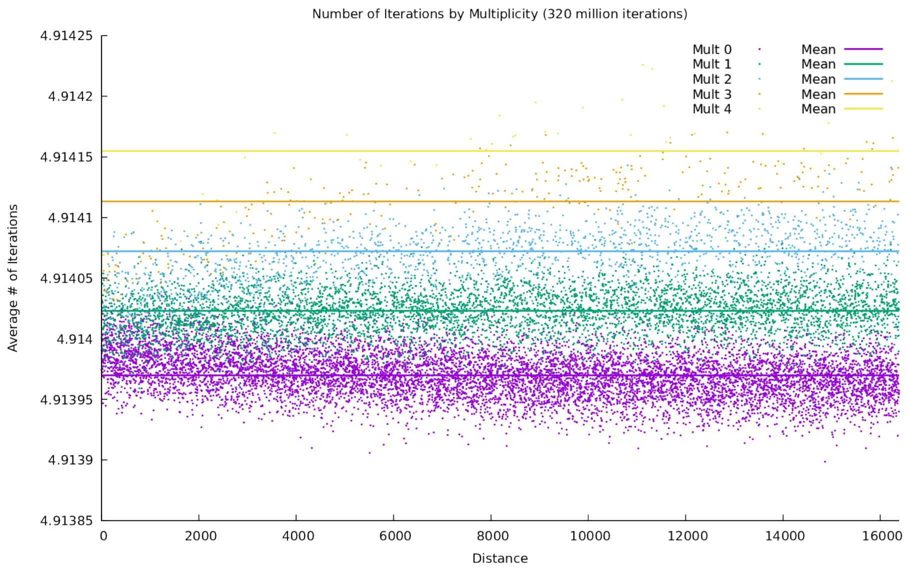


Fig. 5: Attack using the number of decoding iterations against parameters for 256-bit security with fixed threshold decoding.

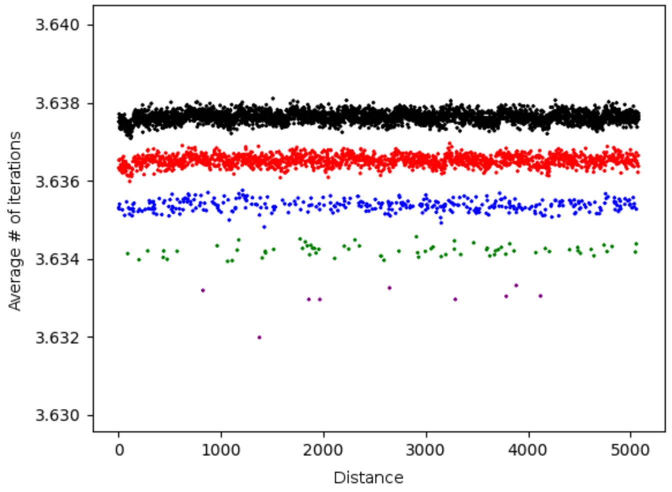


Fig. 6: Attack using the number of decoding iterations, with 2^{25} samples, against parameters for 128-bit security QC-MDPC scheme with variable threshold decoding. The color of the distances indicate their multiplicity in the key spectrum (black $=0$, red $=1$, blue $=2$, green $=3$, purple ≥ 4)

4.1 In-place decoder vs. out-of-place decoder

We observed that changes to the decoding algorithm can have a significant impact on the information gathered during the attack.

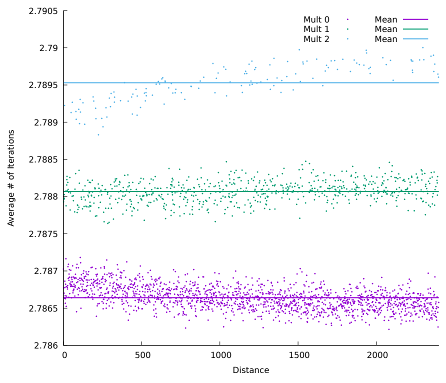
$\mathcal{D}_1$ uses in-place updates to the syndrome which seems to cause some asymmetry in the errors with respect to distance. For example, in Figure 5 the bands converge as distance increases.

Postponing the updates until the end of each iteration (using $\mathcal{B}$ from [27]) seems to eliminate this asymmetry and reduces the correlation between number of iterations and distance multiplicity. This may reduce the efficiency of the attack.

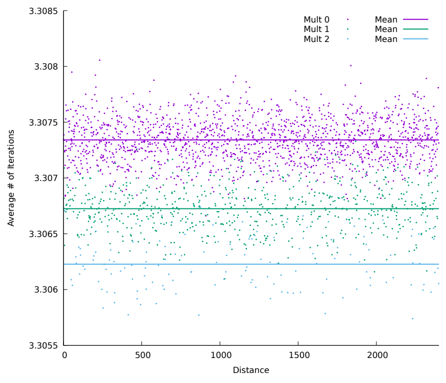
Figure 7 shows a direct comparison between these two types of decoders. Note that the relationship between number of iterations and multiplicity is inverted between decoders.

We are not sure why this is the case but give a possible explanation for the behaviour. When distances match the resulting behaviour is a decrease in total changes to counters (both correct and incorrect). As noted in [20] this decreases the error rate since it decreases the probability of an incorrect change. It also decreases the expected number of bits flipped which could cause an increase in the expected number of iterations.

When multiple bits are flipped at once in the out-of-place decoder the benefit of a correct flip early in an iteration is removed so it is possible that benefit of early flipping is dominated by the increased chance of an incorrect flip.



(a) In-place decoder ($\mathcal{D}_1$).



(b) Out-of-place decoder ($\mathcal{B}$).

Fig. 7: Comparison of in-place and out-of-place decoders with fixed thresholds using 30 million iterations against 80-bit security. Decoder definitions are from [27].

5 Eliminating Decoding Failure Vulnerabilities

In this section we present ParQ — A KEM constructed from repeating a QC-MDPC encryption scheme in order to eliminate the effect of decoding failures. The general idea is for the ciphertext to include several independent encapsulations of the same key in such a way that the scheme achieves CCA2 security, and so that a decapsulation failure occurs in ParQ only if a decryption failure occurs in every instance of the underlying QC-MDPC scheme. As current estimates for the failure rate indicate that failures occur at a rate of roughly 2^{-23}, this suggests that a small amount of parallelization ($3 - 12\times$) will make decapsulation failures occur in ParQ at a negligible rate, thus removing the possibility of implementing a reaction attack based on these failures.

5.1 ParQ — A Parallelized QC-MDPC KEM

ParQ is largely characterized by the same parameters as other QC-MDPC code-based schemes, specifically, k, the plaintext length, $n = 2k$, w, the weight of the secret key, and t, the weight of the error. In addition to these parameters, ParQ has the parameter P, denoting the degree of parallelization. P must be greater than or equal to 2, and should generally be chosen to be in the range of $3 - 12$. ParQ is described by three algorithms: $\mathsf{ParQ.KeyGen}$ for key generation (omitted since it is the same as algorithm 1), $\mathsf{ParQ.Enc}$ for encapsulation, and $\mathsf{ParQ.Dec}$ for decapsulation. It uses three functions which we model as random oracles, ErrGen, PRF, and KDF, which map onto $\mathbb{E}$, $\mathbb{F}_2^k$, and $\{0,1\}^\lambda$, respectively.

Algorithm 9 $\mathsf{ParQ.Enc}$

```
Input: Public key $pk$, a seed $s \in \{0,1\}^k$.
Output: Session key $K$, key encapsulation $C = (c_1, \ldots, c_P)$.
1: for $i = 1$ to $P$ do
2:     Let $e_i = \mathsf{ErrGen}(s||i)$.
3:     Compute $x_i = s \oplus \mathsf{PRF}(e_i||i)$.
4:     Compute $c_i = \mathsf{QCMDPC.Enc}(pk, x_i, e_i)$.
5: Compute $K = \mathsf{KDF}(s)$.
6: Return session key $K$, key encapsulation $C = (c_1, \ldots, c_P)$.
```

5.2 Overview of IND-CCA2 reduction for ParQ

For the rest of this section, we show the IND-CCA2 (INDistinguishable under Chosen Ciphertext Attack) security of the ParQ KEM. We show this by reduction from the OW-CPA (One Way under Chosen Plaintext Attack) security of the QC-MDPC McEliece system. We use the standard definitions of IND-CCA2 and OW-CPA security, which can be found in appendix A for completeness.

Algorithm 10 ParQ.Dec

Input: Secret key sk, public key pk, and encapsulation $C = (c_1, c_2, \ldots, c_P)$.
Output: Session key K, or decapsulation failure symbol $\perp$.

```
1: for $i = 1$ to $P$ do
2:     Run $(x_i, e_i) \leftarrow \mathsf{QCMDPC.Dec}(sk, c_i)$.
3:     if $\mathsf{QCMDPC.Dec}$ succesfully decoded for the first time then
4:         Set used index $j = i$.
5: if $\mathsf{QCMDPC.Dec}$ failed to decode for $i = 1$ to $P$ then
6:     Return decapsulation failure $\perp$.
7: Compute $s = x_j \oplus \mathsf{PRF}(e_j||j)$.
8: Compute $K, C' = (c'_1, c'_2, \ldots, c'_P) \leftarrow \mathsf{ParQ.Enc}(pk, s)$.
9: if $c_i = c'_i$ for all $i \in \{1, \ldots, P\}$ then
10:    Return $K$.
11: else
12:    Return decapsulation failure $\perp$.
```

Theorem 1. *Let $\mathcal{A}$ be an adversary capable of winning the IND-CCA2 security game with the ParQ KEM with q_d decapsulation queries and q_{ErrGen}, q_{PRF}, and q_{KDF} queries to the random oracles* ErrGen*,* PRF*, and* KDF *respectively, in time t and with advantage ϵ. Then there exists a reduction $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine by simulating the IND-CCA2 environment in order to break the OW-CPA security of QC-MDPC McEliece, in time $\approx t$ and with success probability $\gamma(\epsilon/P - \delta)$, where δ is negligible and γ is negligibly close to 1 in the security parameter.*

In order to establish IND-CCA2 security via a reduction from OW-CPA, we need to establish how to embed the given OW challenge c^* into an IND challenge (Section 5.4), and how to successfully respond to decapsulation queries (Section 5.5). Then we need to show that the simulation satisfies several key properties: that the simulated challenge is indistinguishable from a real challenge (Section 5.4), that an adversary's ability to solve the IND challenge allows the simulation to solve the OW challenge (Section 5.4), and that the simulated responses to decapsulation queries are indistinguishable from actual responses to a decapsulation query (Section 5.5).

In ParQ, we have that $e_i = \mathsf{ErrGen}(s||i)$ and $x_i = s \oplus \mathsf{PRF}_i(e_i)$, or $s = x_i \oplus \mathsf{PRF}_i(e_i)$. So for any possible c and i, there is at most one s associated with it such that $c = \mathsf{QCMDPC.Enc}(s \oplus \mathsf{PRF}(\mathsf{ErrGen}(s||i)||i), \mathsf{ErrGen}(s||i))$.

5.3 Simulating the Random Oracle

ParQ makes use of three functions that we will model as random oracles — a pseudo random function PRF, an error generation function ErrGen, and a key derivation function KDF. Each random oracle will be maintained by a standard 'on-the-fly' method. For each oracle, a table is maintained specifying which queries have been made and what the responses were. For each oracle, when a query is made, we first check if it has been queried before, and if so, respond with the same response made before. We then specify how to handle new queries.

For new queries to ErrGen of the form $s||i$ we choose a uniformly random error vector $e \in \mathbb{E}$. We then also calculate $x = s \oplus \mathsf{PRF}(e||i)$ and add e and $c = \mathsf{QCMDPC.Enc}(x, e)$ to the table. We then respond with e.

For new queries to the PRF oracle of the form $e||i$ we first check and see if e is the error vector associated with the challenge ciphertext c^*. We do this by using the generator matrix G to see if $c^* - e$ is a codeword. If so, then we have solved the challenge. Otherwise, generate a uniformly random string from $\{0,1\}^k$, add it to the table and respond.

New queries to KDF can simply be handled by responding with a uniformly random $\{0,1\}^\lambda$.

5.4 Challenge Injection

As we are attempting to solve an OW-CPA challenge, we are given a public key G and a ciphertext c^* and asked to find the (x^*, e^*) such that $c^* = x^* G + e^*$.

To simulate a challenge, we will first select a uniformly random index $j \xleftarrow{\$} \{1, \ldots, P\}$. Then, we will select a uniformly random seed $s \in \{0,1\}^k$. We will run the encapsulation algorithm $\mathsf{ParQ.Enc}$ on the seed s, except that we will *not* query $\mathsf{ErrGen}(s||j)$ to generate e_j, and thus not generate x_j and c_j. Thus we will have $c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_P$ and K.

To finish the challenge encapsulation, we will select a uniformly random bit $b \in \{0,1\}$. If $b = 0$, we will send K, and if $b = 1$ we will send a uniformly random $K' \in \{0,1\}^\lambda$. We will send $C = (c_1, \ldots, c_{j-1}, c^*, c_{j+1}, \ldots, c_P)$ as the encapsulation.

OW Challenge Solution Extraction. We need to show that the adversary's advantage in solving the IND-CCA2 challenge corresponds to an extractor's ability to solve the OW-CPA challenge. Note that the only way for an adversary to distinguish the correct key from an incorrect one is by querying the s associated with each c_i to the KDF oracle. Without having done this, the adversary has no information on K and so she has no advantage in distinguishing a proper K from a random one. Therefore, the adversary's advantage in distinguishing corresponds exactly to their ability to query (and thus find) s.

First, we show that the adversary's probability of querying s to KDF without having queried an e_i for one of the c_i's to PRF (along with i) is negligibly small.

Without having queried some e_i to PRF, the plaintext values $x_1, \ldots, x_p$ provide no information on s. Recall that $s = x_i \oplus \mathsf{PRF}(e_i||i)$. Then $(x_1, \ldots, x_P)$ can be thought of as P maskings of the same value s, with independent masking values. This contains no information about s, unless the adversary has queried at least one e_i to PRF.

Similarly, the values $(e_1, \ldots, e_P)$ provide no information about s, unless the adversary queries $s||i$ to ErrGen for some i. This happens with probability at most $q_{\mathsf{ErrGen}}/2^k$. So as long as s is not queried to ErrGen and $e_i||i$ is not queried to PRF, then both $(x_1, \ldots, x_P)$ and $(e_1, \ldots, e_P)$ give no information about s, and so the encapsulation $C = (c_1, \ldots, c_P)$ does not.

So we have shown that unless the adversary queries $e_i||i$ to PRF or $s||i$ to ErrGen (for any i), the encapsulation $C = (c_1, \ldots, c_P)$ actually contains no information whatsoever about s. Therefore, the adversary can only query random seeds to KDF and so the probability that they query s to KDF is at most $q_{\mathsf{KDF}}/2^k$.

If the adversary queries $e_i||i$ to PRF for any i, then they can easily find s and thus break the indistinguishability challenge. But (as we will establish next), since the adversary has not queried s to KDF or $s||i$ to ErrGen, the adversary has no ability to detect which ciphertext c_i corresponds to the OW challenge c^*. So if the adversary submits an e_i to PRF, with probability $1/P$, this e_i is in fact e^*, and we will solve the OW-CPA challenge.

Indistinguishability of Simulated Challenge. When the adversary is given a challenge encapsulation $C = (c_1, \ldots, c_{j-1}, c^*, c_{j+1}, \ldots, c_P)$, along with a possible key K, we need to ensure that they cannot tell that this is not a correctly formatted encapsulation. Other than replacing c_j with c^*, this is a correct encapsulation. All encapsulations come in the form of P uniform ciphertexts. However a correct encapsulation has the additional property that for each (x_i, e_i) associated with a c_i, $s = x_i \oplus \mathsf{PRF}(e_i||i)$ is the same for all c_i, and $e_i = \mathsf{ErrGen}(s||i)$.

Intuitively, we can see that the only way for an adversary to distinguish between a correctly formatted encapsulation, and one that is generated as in our simulation is by being able to find the (x_i, e_i) associated with at least one of the c_i, and then checking the other c_i through the PRF and ErrGen functions.

Formally, if s has not been queried to *kdf*, $s||i$ has not been queried to ErrGen for any i, and $e_i||i$ has not been queried to PRF for any i, then each x_i and e_i is indistinguishable from being independently and uniformly generated. As such, the ciphertext is perfectly indistinguishable unless the adversary queries $e_i||i$ to PRF for some i. This event also corresponds to the adversary's ability in solving the IND challenge and is considered in the previous subsection.

5.5 Simulating Decapsulation Queries

When we receive a query for decapsulation $C = (c_1, \ldots, c_P)$, we need to respond with the decapsulation K. Upon receiving the query, we lookup the ErrGen table for P queries of the form $s||1, s||2, \ldots, s||P$ such that c_i is in the table for each $s||i$. If such a set of P queries is found, we respond with $\mathsf{KDF}(s)$. Otherwise, we return the decryption failure symbol $\perp$.

We must establish that this simulation is indistinguishable from a real decapsulation oracle. To establish this, we need to show two things: that we do not respond with $\perp$ when we should respond with a decapsulated key, and that we do not respond with a decapsulated key when we should respond with $\perp$. For the first point, we must ensure that any potential encapsulation query made by the adversary in any way *other* than by beginning with a seed s and generating each c_i according to $c_i = \mathsf{QCMDPC.Enc}(s \oplus \mathsf{PRF}(\mathsf{ErrGen}(s||i)||i), \mathsf{ErrGen}(s||i))$ only results in a ciphertext that would not return $\perp$ with negligible probability.

As previously noted, any ciphertext and index pair (c, i) is associated with exactly one seed s induced by $s = x \oplus \mathsf{PRF}(e||i)$, as there is at most one pair (x, e) associated with c. For the ciphertext to be valid (and thus for a decapsulation oracle to *not* output $\perp$), it must be the case that $\mathsf{ErrGen}(s||i) = e$. So for a decapsulation query $C = (c_1, \ldots, c_P)$, for a correct decapsulation oracle to not return $\perp$, each c_i must be associated with the same seed s, and for each i, $e_i = \mathsf{ErrGen}(s||i)$.

When an adversary submits a decapsulation query, if it is not the case that a single s has been queried P times to ErrGen in the form $s||1, s||2, \ldots, s||P$, then there are two possibilities. Either for at least one c_i, no query has been made of the form $s'||i$ that results in c_i, or such a query has been made but the s' is different from one other s.

In the latter case, our simulation would return $\perp$, and indeed this is consistent with what an actual decapsulation oracle would return, as each c_i is not associated with the same seed, which the decapsulation algorithm can always detect.

In the first case, where $s||i$ has not been queried to generate c_i, our simulation will return $\perp$. This is usually consistent with what a correct decapsulation oracle will return. The only case an inconsistency would arise is if, when $\mathsf{ErrGen}(s||i)$ is later queried, $\mathsf{ErrGen}(s||i) = e_i$, despite it not having been queried at the time that the decapsulation query is made. As ErrGen is a random oracle, this only happens with probability at most $1/\#\mathbb{E}$.

Showing that we do not respond with a decapsulation when we should respond with $\perp$ corresponds to the fact that we will never have a decoding failure. In a real decapsulation oracle, if a decoding error were to occur for each c_i, then we would be forced to respond with $\perp$. But in our simulated version, we would respond with the correct decapsulation, as we would have seen it from the random oracle. However, because of the parallelization, we can see that any s will result in errors that will give a total decapsulation failure (i.e. the decoding procedure fail for all $e_1, e_2, \ldots, e_P$) with probability ζ^P, where ζ is the decoding failure rate. Given this, we need to consider the probability that an adversary queries an encapsulation C that should result in a decapsulation failure. We should note that it should be hard to identify error vectors which will result in decoding failures (or else an adversary may not need to launch the GJS attack at all), but as we have no proof of this, we assume an adversary can perfectly distinguish which error vectors will result in decoding failures.

A fraction ζ^P of seeds will result in an encapsulation that cannot be decapsulated. So in q_{ErrGen} queries to the random oracle, the probability that the adversary is able to find such a seed is less than $q_{\mathsf{ErrGen}}\zeta^P$. We assume that P is chosen so that this quantity is negligible (we discuss this further in Section 5.7).

5.6 Combining

We let Game 0 (or $G0$) refer to the original IND-CCA2 game. We let Game 1 ($G1$) refer to the simulated IND-CCA2 game, where the challenge and decapsulation oracle are simulated.

To simplify our calculation, we also define three events that can occur in the process of either Game 0 or Game 1.

– Event 1 (or $E1$) refers to the event that the adversary $\mathcal{A}$ queries s to the KDF oracle.
– Event 2 (or $E2$) refers to the event of the adversary $\mathcal{A}$ querying one of the $e_i||i$ (from the challenge encapsulation) to PRF prior to querying s to KDF or $s||i$ to ErrGen.
– Event 3 (or $E3$) is the event that the adversary $\mathcal{A}$ breaks the distinguishability of the simulated decapsulation oracle. Specifically, that they query an $s||i$ to ErrGen such that $\mathsf{ErrGen}(s||i)$ will result in a decoding failure for each i, or that they submit a ciphertext to the decapsulation oracle without querying the associated s to construct it, and that when s is later queried, it does result in the proper error vector, and that they do this prior to Event 1 or 2.

Then, according to the discussion in Sections 5.4 and 5.5, we perform the following calculation:

$$\begin{aligned}\frac{1}{2}+\epsilon =&\Pr_{G0}[\mathcal{A}\text{ wins}]\\ \le&\Pr_{G0}[\mathcal{A}\text{ wins}|\neg E1]+\Pr_{G0}[E1]\le\frac{1}{2}+\Pr_{G0}[E1].\end{aligned}\tag{2}$$

This tells us that $\epsilon \le \Pr_{G0}[E1]$. Next, we consider $\Pr_{G0}[E1]$:

$$\epsilon\le\Pr_{G0}[E1]\le\Pr_{G0}[E2]+\Pr_{G0}[E1|\neg E2]\le\Pr_{G0}[E2]+\frac{q_{\mathsf{KDF}}+q_{\mathsf{ErrGen}}}{2^k}.\tag{3}$$

Next, we relate $\Pr_{G0}[E2]$ to $\Pr_{G1}[E2]$. This is done simply by noting that

$$\Pr_{G0}[E2]\le\Pr_{G0}[E2|\neg E3]+\Pr_{G0}[E3],\tag{4}$$

and that

$$\Pr_{G0}[E2|\neg E3]=\Pr_{G1}[E2|\neg E3],\qquad\qquad \Pr_{G0}[E3]=\Pr_{G1}[E3].\tag{5}$$

Then finally, noting that our ability to solve the OW-CPA challenge corresponds to $1/P$ times $\Pr_{G1}[E2\wedge\neg E3]$, we get that

$$\begin{aligned}\Pr[\text{We win OW-CPA game}]\ge&\frac{1}{P}\Pr_{G1}[E2\wedge\neg E3]\\ =\frac{1}{P}\Pr_{G0}[\neg E3]\Pr_{G0}[E2|\neg E3]\ge&\frac{1}{P}\Pr_{G0}[\neg E3]\left(\Pr_{G0}[E2]-\Pr_{G0}[E3]\right),\end{aligned}\tag{6}$$

and so

$$\Pr[\text{We win OW-CPA game}]\ge\frac{\gamma}{P}(\epsilon-\delta),\tag{7}$$

where

$$\delta = \frac{q_{\mathsf{ParQ.Dec}}}{\#\mathbb{E}} + \frac{q_{\mathsf{KDF}} + q_{\mathsf{ErrGen}}}{2^k} + q_{\mathsf{ErrGen}}\zeta^P \tag{8}$$

and

$$\gamma = 1 - \frac{q_{\mathsf{ParQ.Dec}}}{\#\mathbb{E}} - q_{\mathsf{ErrGen}}\zeta^P. \tag{9}$$

5.7 Comparison

In this section we compare aspects of ParQ's efficiency and security with other code-based KEMs, many of which have been submitted to NIST's Post-Quantum Cryptography project [1]. We restrict ourselves to code-based systems for direct comparison. Comparing code-based systems to other post-quantum systems has been done elsewhere in the literature, for example in [5]. All comparisons are done considering parameters that have been proposed for 128 bits of post-quantum security, or NIST's security level 5 (AES 256) (see table 1).

Scheme	Public Key	Secret Key	Encapsulation	Static Key Use
CAKE[5]	8193	8193	8225	✗
BIKE-1[2]	8188	8188	8188	✗
BIKE-2[2]	4094	8188	4094	✗
BIKE-3[2]	9033	9033	9033	✗
Classic McEliece[6]	1 357 824	14 080	240	✓
ParQ	4094	8193	98 313	✓

Table 1: Length in bytes of keys and encapsulations for Code-Based KEMs. Using 8192128 for Classic McEliece

While we do not have specific data on the speed of ParQ as it compares to other systems, one can expect that, because it requires P encapsulations and the decapsulation must be constant time to avoid side-channel timing attacks, the time to encapsulate and decapsulate likely increases by a factor of roughly P as opposed to a construction like CAKE.

Here we have selected the parameter P to be 12. This reflects the fact that it reduces the decapsulation error rate to be on the order of 2^{-252}, presumably hard for even a fully quantum adversary to find a seed that results in a total decapsulation failure (even if the adversary is perfectly able to tell which errors will result in decoding failures, which is presumably hard without the secret key). One could choose P to be much lower, on the order of 2 or 3. While it appears that the GJS attack would be mitigated by these low values of P, (increasing the attack complexity to an estimated 2^{66} or 2^{87} queries respectively), decapsulation errors may still occur in the normal lifetime of a key, meaning that the guarantees of the CCA2 security proof would not apply. While we have specified that P must be at least 2, note that P could be set to 1. This would cause the scheme to bear some resemblance to the CAKE scheme [5] or the BIKE-1 scheme [2]. However, this would cause the scheme to be vulnerable to the GJS attack, which is why these schemes currently insist on using the public key ephemerally.

6 Conclusion and Future Work

We have explored and answered several fundamental questions that arose as a result of the powerful GJS reaction attack on QC-MDPC McEliece. We analyzed the origin of this leak: a bias on the distribution of the syndrome weight. This analysis allows a better understanding of the GJS attack and we deduce other side-channel attacks exploiting all decoding instances.

Our analysis provides quantitative bounds on the minimal number of samples needed to deduce relevant information (using Chernoff's bound), which could be used to deduce better parameters to prevent attacks on the syndrome weight. Other side-channel attacks on different (noisier) parameters exploiting the same idea will be even more costly.

We also discussed how variations in the implemented decoding procedure can affect the attack. Lastly we have showed how decoding failures can be addressed at the protocol level by constructing a KEM that entirely defeats the GJS reaction attack for QC-MDPC, without altering the parameters of the system. We provided a proof of the CCA2 security of the KEM in the random oracle model. Notably, this proof considered the possibility of decoding failures, meaning that it should not be possible to attack the system by exploiting decoding failures.

The security of ParQ is proven in the random-oracle model. A complete and thorough analysis of post-quantum security would require a security reduction in the quantum random-oracle model [8]. Showing that ParQ (or a small modification of ParQ) is secure in this model would give greater post-quantum assurance.

MDPC codes are still a recent proposal. Even though they are close to the thoroughly studied LDPC codes, they seem to behave differently, in particular as far as decoding is concerned [9]. It is very likely that the state of the art for decoding MDPC codes will evolve quickly, especially considering the NIST call for quantum safe primitives. Interestingly, it seems that more efficient decoders (*e.g.* those using variable threshold rules) are more prone to information leakage, and thus better decoders might not be safer. Evaluating new decoding algorithm, their failure rates and running time distribution with respect to this work could indicate whether and at what cost QC-MDPC codes could be used for PKEs as safely as for KEMs.

References

1. NIST post-quantum cryptography project, round 1 submissions (2017), `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`
2. Aragon, N., Barreto, P.S.L.M., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Güneysu, T., Melchor, C.A., Misoczki, R., Persichetti,

E., Sendrier, N., Tillich, J.P., Zémor, G.: BIKE — bit flipping key encapsulation (2017), http://bikesuite.org
3. Augot, D., Batina, L., Bernstein, D.J., Bos, J., Buchmann, J., Castryck, W., Dunkelman, O., Güneysu, T., Gueron, S., Hülsing, A., Lange, T., Mohamed, M.S.E., Rechberger, C., Schwabe, P., Sendrier, N., Vercauteren, F., Yang, B.Y.: Initial recommendations of long-term secure post-quantum systems (2015), http://pqcrypto.eu.org/docs/initial-recommendations.pdf
4. Baldi, M., Bodrato, M., Chiaraluce, F.: A new analysis of the McEliece cryptosystem based on QC-LDPC codes. In: Proceedings of Security and Cryptography for Networks, 6th International Conference (SCN 2008). pp. 246–262 (2008)
5. Barreto, P.S.L.M., Gueron, S., Güneysu, T., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.P.: CAKE: Code-based algorithm for key encapsulation. Cryptology ePrint Archive, Report 2017/757 (2017)
6. Bernstein, D.J., Chou, T., Lange, T., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Wang, W.: Classic mceliece (2017), https://classic.mceliece.org
7. Bernstein, D.J., Chou, T., Schwabe, P.: McBits: Fast constant-time code-based cryptography. In: CHES 2013. LNCS, vol. 8086, pp. 250–272. Springer (2013)
8. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Asiacrypt 2011. LNCS, vol. 7073, pp. 41–69. Springer (2011)
9. Chaulet, J.: Étude de cryptosystèmes à clé publique basés sur les codes MDPC quasi-cycliques. Ph.D. thesis, Université Pierre et Marie Curie-Paris VI (2017)
10. Chaulet, J., Sendrier, N.: Worst case QC-MDPC decoder for mceliece cryptosystem. In: IEEE International Symposium on Information Theory, (ISIT 2016). pp. 1366–1370 (2016)
11. Chen, C., Eisenbarth, T., von Maurich, I., Steinwandt, R.: Differential power analysis of a mceliece cryptosystem. In: 13th International Conference on Applied Cryptography and Network Security (ACNS 2015) Revised Selected Papers. pp. 538–556 (2015)
12. Chen, C., Eisenbarth, T., von Maurich, I., Steinwandt, R.: Horizontal and vertical side channel analysis of a mceliece cryptosystem. IEEE Trans. Information Forensics and Security 11(6), 1093–1105 (2016)
13. Chou, T.: QcBits: Constant-time small-key code-based cryptography. In: CHES 2016. LNCS, vol. 9813, pp. 280–300. Springer (2016)
14. Deneuville, J.C., Gaborit, P., Zémor, G.: Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In: PQCrypto 2017. LNCS, vol. 10346, pp. 18–34. Springer (2017)
15. Dent, A.W.: A designer's guide to KEMs. In: Cryptography and Coding: 9th IMA International Conference, Proceedings. LNCS, vol. 2898, pp. 133–151. Springer (2003)
16. Döttling, N., Dowsley, R., Müller-Quade, J., Nascimento, A.C.A.: A CCA2 secure variant of the McEliece cryptosystem. IEEE Transactions on Information Theory 58(10), 6672–6680 (2012)
17. Fabšič, T., Hromada, V., Stankovski, P., Zajac, P., Guo, Q., Johansson, T.: A reaction attack on the QC-LDPC McEliece cryptosystem. In: PQCrypto 2017. LNCS, vol. 10346, pp. 51–68. Springer (2017)
18. Gaborit, P.: Shorter keys for code based cryptography. In: Proceedings of WCC 2005. pp. 81–90 (2005)
19. Gallager, R.G.: Low-Density Parity-Check Codes. Ph.D. thesis, Massachusetts Institute of Technology (1963)

20. Guo, Q., Johansson, T., Stankovski, P.: A key recovery attack on MDPC with CCA security using decoding errors. In: Asiacrypt 2016. LNCS, vol. 10031, pp. 789–815. Springer (2016)
21. Habib, M., McDiarmid, C., Ramirez-Alfonsin, J., Reed, B.: Probabilistic methods for algorithmic discrete mathematics, vol. 16. Springer Science & Business Media (2013)
22. Heyse, S., von Maurich, I., Güneysu, T.: Smaller keys for code-based cryptography: QC-MDPC McEliece implementations on embedded devices. In: Proceedings of the 15th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2013). pp. 273–292 (2013)
23. Kobara, K., Imai, H.: Semantically secure McEliece public-key cryptosystems-conversions for McEliece PKC. In: Public Key Cryptography 2001. pp. 19–35. PKC '01, Springer-Verlag (2001)
24. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: CRYPTO '96. LNCS, vol. 1109, pp. 104–113. Springer (1996)
25. von Maurich, I., Güneysu, T.: Towards side-channel resistant implementations of QC-MDPC McEliece encryption on constrained devices. In: PQCrypto 2014. LNCS, vol. 8772, pp. 266–282. Springer (2014)
26. von Maurich, I., Heberle, L., Güneysu, T.: IND-CCA secure hybrid encryption from QC-MDPC Niederreiter. In: PQCrypto 2016. LNCS, vol. 9606, pp. 1–17. Springer (2016)
27. von Maurich, I., Oder, T., Güneysu, T.: Implementing QC-MDPC McEliece encryption. ACM Transactions on Embedded Computing Systems (TECS) 14(3), 44:1–44:27 (2015)
28. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. Deep Space Network Progress Report 44, 114–116 (1978)
29. Misoczki, R., Tillich, J.P., Sendrier, N., Barreto, P.S.L.M.: MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In: 2013 IEEE International Symposium on Information Theory. pp. 2069–2073 (2013)
30. Monico, C., Rosenthal, J., Shokrollahi, A.: Using low density parity check codes in the McEliece cryptosystem. In: IEEE International Symposium on Information Theory – ISIT'2000. p. 215. IEEE (2000)
31. Niederreiter, H.: Knapsack type of cryptosystems and algebraic coding theory 15, 19–34 (1986)
32. Rosen, A., Segev, G.: Chosen-ciphertext security via correlated products. In: TCC 2009. LNCS, vol. 5444, pp. 419–436. Springer (2009)
33. Strenzke, F.: A timing attack against the secret permutation in the McEliece PKC. In: PQCrypto 2010. LNCS, vol. 6061, pp. 95–107. Springer (2010)
34. Strenzke, F.: Timing attacks against the syndrome inversion in code-based cryptosystems. In: PQCrypto 2013. LNCS, vol. 7932, pp. 217–230. Springer (2013)
35. Strenzke, F., Tews, E., Molter, H.G., Overbeck, R., Shoufan, A.: Side channels in the McEliece PKC. In: PQCrypto 2008. LNCS, vol. 5299, pp. 216–229. Springer (2008)
36. Yoshida, Y., Morozov, K., Tanaka, K.: Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In: PQCrypto 2017. LNCS, vol. 10346, pp. 35–50. Springer (2017)

A Security Definitions and Games

These standard definitions, used in the security proof for ParQ, have been replicated from [15] for the sake of completeness.

The IND-CCA2 and OW-CPA games take place between two parties, the challenger $\mathcal{C}$, and the attacker or adversary, $\mathcal{A}$.

Game 1 (IND-CCA2 Challenge).

1. $\mathcal{C}$ obtains $(pk, sk) \leftarrow \mathsf{ParQ.KeyGen}(1^\lambda)$, and sends pk to $\mathcal{A}$. $\mathcal{C}$ runs $\mathsf{ParQ.Enc}(s)$ with a uniformly random s, obtaining K_0, C. $\mathcal{C}$ then generates a uniformly random $K_1 \in \{0,1\}^\lambda$, and a uniformly random bit $b \in \{0,1\}$. $\mathcal{C}$ then sends C and K_b to $\mathcal{A}$.
2. $\mathcal{A}$ may freely send decapsulation queries C to $\mathcal{C}$. $\mathcal{C}$ responds by sending $\mathsf{ParQ.Dec}(C)$ to $\mathcal{A}$. The only exception is that $\mathcal{A}$ may not send the challenge encapsulation C as a decapsulation query.
3. Eventually, $\mathcal{A}$ must return a bit b' as a guess for the bit b. $\mathcal{A}$ is said to have won the IND-CCA2 game if $b' = b$.

We write $\mathcal{A}$'s ability to win Game 1 as $1/2 + \epsilon$. We call ϵ the adversary's *advantage* in breaking IND-CCA2 security.

Game 2 (OW-CPA Challenge).

1. $\mathcal{C}$ generates $(pk, sk) \leftarrow \mathsf{QCMDPC.KeyGen}(1^\lambda)$. They select a uniformly random $x \xleftarrow{\$} \{0,1\}^k$ and $e \xleftarrow{\$} \{0,1\}^n$, with e having weight t. They then compute $c^* \leftarrow \mathsf{QCMDPC.Enc}(pk, x, e)$ and sends c^* and pk to $\mathcal{A}$.
2. $\mathcal{A}$ performs some computation on c^* and pk. Eventually they must produce an x'. $\mathcal{A}$ is said to have won the OW-CPA game if $x' = x$.

B Choosing the Bit-flipping Thresholds

In standard literature, rules for threshold computation are heuristic and are not available for all parameter sets. To convince that our experiments were fair we describe the rules we used for fixed and variable threshold. We denote $d = w/2$ the column weight.

Monitoring Strategy: For a given set of parameters, we run the bit-flipping algorithm on many random instances and we choose at each iteration the threshold which minimizes the error weight at the end of all flips[1]. This is possible in a simulation because we know the initial error pattern and we can monitor its evolution. We will refer to this as the "monitoring strategy" and use it as a tool to define the thresholds.

Fixed Thresholds: For a given set of parameters, we run a simulation using the monitoring strategy and we keep track of the threshold values used at the first iteration. The maximum of those values is kept as the fixed threshold, say b_0, for the first iteration. We run a second simulation, for which the first threshold is fixed to b_0 and the monitoring strategy is used for the following iterations. We keep track of the threshold values used at the second iteration. The maximum of those values is kept as the fixed threshold, say b_1, for the second iteration. We repeat this until we reach the maximal expected number of iterations.

[1] In case of a tie, we choose the smallest threshold, but never smaller than $d/2$.

Variable thresholds: For a given set of parameters, the goal here is to establish a rule $b_i(\sigma)$, $i \geq 0$, giving the i-th iteration threshold as a function of the syndrome weight σ. Assuming all b_ℓ for $\ell < i$ are known, we run a simulation using the functions $b_0, \dots, b_{i-1}$ for the first i iterations and using the monitoring strategy after that. We keep track of the pairs (σ, b) of syndrome weights and threshold values used at the i-th iteration. For each syndrome weight σ, we define $f_i(\sigma)$ as the average of all thresholds observed. Next, using the least square method, we find the quadratic[2] function $g_i(\sigma)$ which best approximates all the $(\sigma, f_i(\sigma))$ where each $(\sigma, f_i(\sigma))$ is weighted by the number of occurrences of the syndrome weight σ. The threshold function for the i-th iteration will be $\lceil g_i(\sigma) \rceil$. We add the condition that b_i is increasing with σ and we get $b_i : \sigma \to \max\left(b_i^{\min}, \lceil g_i(\sigma) \rceil\right)$ where $b_i^{\min}$ is the minimal value of $\lceil g_i(\sigma) \rceil$ over the observed range for σ, and is never smaller than $d/2$.

Results and Comments. We give below the threshold rules we used for our simulations deduced from the above-mentioned process. Note that we do not claim, nor observed, that those rules are giving any kind of improvement in speed or failure rate.

Fixed Thresholds. For 80-bit security parameters, $(k, w, t) = (4801, 90, 84)$, we have $(b_i)_{i \geq 0} = (30, 28, 26, 25, 23, \dots)$. The dots meaning that the last value is repeated as much as necessary. We remark that, for the same parameters, QcBits [13] uses thresholds that are exactly one unit lower for the first 4 iterations. This probably reflects the fact that our strategy is rather conservative.

For 128-bit security, $(k, w, t) = (10163, 142, 134)$, we get $(b_i)_{i \geq 0} = (46, 43, 41, 40, 39, 37, 36, \dots)$. Finally for 256-bit security, $(k, w, t) = (32771, 274, 264)$ we obtain $(b_i)_{i \geq 0} = (83, 80, 77, 74, 72, \dots)$.

Variable Thresholds.

$$\begin{array}{c}(k,w,t) = \\ (4801, 90, 84)\end{array} \Rightarrow \begin{cases} b_0(\sigma) = \lceil 11.1 + 0.00919\,\sigma \rceil \\ b_1(\sigma) = \max(24, \lceil 38.7 - 0.0242\sigma + 1.004\,10^{-5}\sigma^2 \rceil) \\ b_i(\sigma) = \max(24, \lceil 34.9 - 0.0195\sigma + 0.836\,10^{-5}\sigma^2 \rceil), i \geq 2, \end{cases}$$

$$\begin{array}{c}(k,w,t) = \\ (10163, 142, 134)\end{array} \Rightarrow \begin{cases} b_0(\sigma) = \lceil 15.5 + 0.00665\,\sigma \rceil \\ b_1(\sigma) = \lceil 51.7 - 0.0128\,\sigma + 0.257\,10^{-5}\sigma^2 \rceil \\ b_i(\sigma) = \max(37, \lceil 40.1 - 0.00395\,\sigma + 9.50\,10^{-7}\sigma^2 \rceil, i \geq 2 \end{cases}$$

$$\begin{array}{c}(k,w,t) = \\ (32771, 274, 264)\end{array} \Rightarrow \begin{cases} b_0(\sigma) = \lceil 22.9 + 0.00402\,\sigma \rceil \\ b_1(\sigma) = \lceil 18.2 + 0.00431\,\sigma \rceil \\ b_2(\sigma) = \max(71, \lceil 315.8 - 0.0422\,\sigma + 0.182\,10^{-5}\sigma^2 \rceil) \\ b_i(\sigma) = \max(69, \lceil 62.5 + 0.000648\,\sigma \rceil), i \geq 3. \end{cases}$$

[2] We use the linear approximation unless the quadratic approximation gives different values of $b_i(\sigma) = \lceil g_i(\sigma) \rceil$ for σ in the observed range.