QC-MDPC KEM: A Key Encapsulation Mechanism Based on the QC-MDPC McEliece Encryption Scheme

Principal submitter:

Atsushi Yamada
Atsushi.Yamada@isara.com
+1 877 319-8576
ISARA Corporation
560 Westmount Rd North,
Waterloo, ON, Canada. N2L 0A9

Auxiliary submitters:

Edward Eaton
Kassem Kalach
Philip Lafrance
Alex Parent

Owner: ISARA Corporation
Developer: Alex Parent

Atsushi Yamada

Atsushi Yamada

November 30 2017

Date

Abstract

This document constitutes the algorithmic specifications and other supporting documentation regarding an IND-CPA secure ephemeral Key Encapsulation Mechanism (KEM) based on the Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) McEliece encryption scheme. This report is one part of a submission to the NIST Post-Quantum Cryptography Project. In particular, this report proposes the KEM described herein for NIST standardization.

Table of Contents

1. Preliminaries

This document profiles the Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) McEliece encryption scheme and specifies a simple, efficient, and secure Key Encapsulation Mechanism (KEM) which utilizes it. In particular, this document proposes the aforementioned KEM for NIST post-quantum standardization. The described encryption scheme is already well known and its security against classical adversaries has been diligently studied by experts world wide. Moreover, its security against quantum capable adversaries has been a major focus of study. This document includes an extensive post-quantum security analysis of both the QC-MDPC encryption scheme and the associated KEM.

This chapter serves to give the relevant background information on QC-MDPC McEliece to facilitate understanding of later chapters. We begin by giving a history of the McEliece encryption scheme.

In 1978 Robert J. McEliece published his seminal paper "A public-key cryptosystem based on algebraic coding theory" [30] wherein he succinctly described a secure encryption scheme based on binary Goppa codes (c.f [31, Chapter 5]). McElieces' paper was published less than a year after the famed RSA paper [42]. This is interesting for a few reasons; not the least of which are the following. Efficiency of the original McEliece system aside, the concrete security of the scheme has been available for study for nearly forty years and in that time the scheme has not been broken in the classical setting. Moreover, over this time a high level of confidence in the scheme's security against a quantum adversary has been grown (a rigorous analysis of the scheme against a quantum capable adversary using state-of-the-art attacks is given in Chapter 2).

The "test of time" is of paramount importance for the widespread adoption of any cryptographic protocol. For example, lattice-based, and supersingular isogeny-based cryptography are relatively young, and so general understanding of the fundamental security of such schemes is immature and is only studied by a small subset of those who study such things. While this does

not directly imply any security vulnerabilities in those younger branches of cryptography, the more conservative approach is to implement time-tested algorithms.

Multivariate polynomial-based cryptography is slightly older than lattice-based and isogeny-based cryptography. The original 1988 scheme by Matsumoto and Imai [27] was broken in 1998 [36], and many subsequent multivariate schemes have also been broken. Not all multivariate schemes are broken, for example Ding and Schmidt's scheme Rainbow is believed to be quantum-secure (for example see [38]). It should be noted however, that Rainbow is a signature scheme and that this document proposes a KEM.

Although the "hard problems" underlying the security of the original McEliece cryptosystem have not received the same level of attention as say integer factorization or discrete logarithms, it has still been studied extensively and confidence in their difficulty remains high.

It is true that an unsettling number of code-based schemes have been broken [45, 15, 16, 32, 12]. However, Misoczki in his PhD thesis [33] addresses this problem and argues in essence that this is perhaps not as important a concern as some believe. Misoczki argues that the broken McEliece variants are all based on *algebraic codes* and that the inherent algebraic structure of such codes is what so often leads to their insecurity. It is partly for this reason that Misoczki et. al. published their 2012 paper "MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes" [34]. The new schemes are based on *graph-based codes* which do not have exploitable algebraic structures (as argued by Misoczki). The original proposal by McEliece did not use graph-based codes, but rather a type of algebraic code called (binary) Goppa Codes. It is concluded then that binary Goppa codes, while still considered secure, do not seem to be the optimal choice for security. It is one of the schemes from [34] that this document uses as the work-horse within the proposed KEM.

The reader may be aware of a recent key recovery attack on QC-MDPC [19]. This attack uses information gained from decoding failures to reconstruct the secret key. This attack is completely defeated by avoiding the use of static keys. For this reason, it is suggested that the KEM proposed in this document be used in an exclusively ephemeral context. However, as discussed in Section 4.3, the attack requires a large number of decoding failures, and so the accidental reuse of an ephemeral key does not necessarily spell disaster.

Below we give the relevant definitions needed to understand the cryptosystems described herein.

1.1 Definitions and Notation

In the following n is a positive integer and $\mathbb{F}_2^n$ is the finite field of 2^n elements. We write $\log(\cdot)$ to denote the base-2 logarithm, and $\oplus$ to denote the bit-wise exclusive or operation.

Definition 1.1.1 (Hamming weight)**.** *Let $x = (x_0, x_1, \ldots, x_{n-1}) \in \mathbb{F}_2^n$ be a binary vector. Then the* Hamming weight *of a x, denoted $\mathbf{wt}(x)$, is given by $\mathbf{wt}(x) = \sum_{i=0}^{n-1} x_i$. Equivalently it is the number of non-zero components in the vector.*

In this document we will refer to the Hamming weight simply as *weight*.

Definition 1.1.2 (Linear Map)**.** *Let A, and B be vector spaces over $\mathbb{F}_2^n$. A function $f : A \to B$ is a linear map iff for all $x, y \in A$ and for all $c \in \mathbb{F}_2$:*

1. *$f(x + y) = f(x) + f(y)$, and*
2. *$f(cx) = cf(x)$.*

A more general definition allows for A and B to be vector spaces over arbitrary fields rather than $\mathbb{F}_2^n$, but this definition is suitable for the needs of this document. These linearity properties allow us to discuss linear codes in the language of vector (sub)spaces as follows.

Definition 1.1.3 (Linear Code)**.** *An (n, k)-linear code $\mathcal{C}$ is a vector subspace of $\mathbb{F}_2^n$ such that $|\mathcal{C}| = 2^k$.*

Vectors $c \in \mathcal{C}$ are referred to as *codewords*, whereas we may refer to arbitrary vectors in $\mathbb{F}_2^n$ simply as *words*.

Definition 1.1.4 (Distance)**.** *For some linear code $\mathcal{C}$ we can define a distance metric $d : \mathcal{C} \times \mathcal{C} \to \mathbb{Z}_{\geq 0}$ as*

$$d(u, v) = \mathbf{wt}(u - v).$$

Definition 1.1.5 (Minimum Distance)**.** *The* minimum distance *of a code $\mathcal{C}$, denoted d_0, is defined as*

$$d_0 = \min\{\mathbf{wt}(c) \mid c \in \mathcal{C} - \{0\}\}.$$

Equivalently, this value may be defined as $d_0 = \min\{\mathbf{wt}(u - v) \mid u, v \in \mathcal{C}, u \neq v\}$.

Definition 1.1.6 (Code Rate)**.** *The* code rate *is defined as $R = k/n$. This value represents the proportion between the bits of codewords that are useful and those that are redundant or noise. A higher code rate implies better error detection and correction.*

Because linear codes are themselves vector spaces, they can be generated by a basis. The most convenient representation of such a basis is in matrix form as follows.

Definition 1.1.7 (Generator Matrix)**.** *A matrix $G \in \mathbb{F}_2^{k\times n}$ is a* generator matrix *for an (n,k)-linear code $\mathcal{C} \subseteq \mathbb{F}_2^n$ iff*

$$\mathcal{C} = \{mG \mid m \in \mathbb{F}_2^k\}.$$

Definition 1.1.8 (Parity-check Matrix)**.** *A matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ is a* parity-check matrix *for an (n,k)-linear code $\mathcal{C} \subseteq \mathbb{F}_2^n$ iff*

$$\mathcal{C} = \{c \in \mathbb{F}_2^n \mid Hc^T = 0\}.$$

Definition 1.1.9 (Syndrome)**.** *The* syndrome *s of a vector $c \in \mathbb{F}_2^n$ with respect to a parity-check matrix $H \in \mathbb{F}_2^{r\times n}$ is given by $s^T = Hc^T \in \mathbb{F}_2^r$.*

It follows immediately from the definition of parity-check matrices that if H is a parity-check matrix for code $\mathcal{C}$, then all codewords $c \in \mathcal{C}$ have a syndrome of 0. Since each row of G is itself a codeword it follows that $HG^T = 0$. This fact is useful because it allows one to efficiently compute a generator matrix from a parity-check matrix (c.f. Section 2.1.1).

Given an (n,k)-linear code, the value n is usually referred to as the *length* of the code, and k is referred to as the *dimension*. In what follows, the value $r = (n-k)$ is referred to as the *co-dimension* of the code.

Definition 1.1.10 (Moderate Density Parity-Check (MDPC) code)**.** *An (n,r,w)-MDPC code is a linear code of length n and co-dimension r whose parity-check matrix has a constant weight $w \in O\left(\sqrt{n\log(n)}\right)$.*

Definition 1.1.11 (Quasi-Cyclic code)**.** *A linear code $\mathcal{C} \subseteq \mathbb{F}_2^n$ is* quasi-cyclic *if there exists a positive integer $n_0 \in \{1,2,\ldots,n-1\}$[1] such that for every codeword $c \in \mathcal{C}$ the word c' obtained from a right cyclic shift of c by n_0 positions is itself a codeword of $\mathcal{C}$.*

This brings us to our final definition of this section.

Definition 1.1.12 (QC-MDPC code)**.** *An (n,r,w)-linear code is a* Quasi-Cyclic Moderate Density Parity-Check (QC-MDPC) code *if it is both an MDPC code and a Quasi-Cyclic code.*

Now we make some important remarks.

Remark 1.1.1. *When $n = n_0 r$ for some positive integer r, then it is possible to construct both the parity-check matrix and the generator matrix so that they are composed of square, $r \times r$ circulant blocks. Hence, exactly one row (or column) from each circulant block is needed to be stored in order to describe the matrices in their entirety.*

Remark 1.1.2. *The algebra of $r \times r$ circulant matrices is isomorphic to that of polynomials over $\mathbb{F}_2[x]/\langle x^r - 1\rangle$, the ring of polynomials modulo $x^r - 1$ over*

[1] Most articles omit this constraint on n_0. However, we contend that this constraint is important or else, taking $n_0 = n$ implies that every code is Quasi-Cyclic.

$\mathbb{F}_2$. *This isomorphism allows for efficient computations because, it allows one to use efficient polynomial multiplication instead of more bulky matrix multiplication algorithms within the protocols.*

The remainder of this document is organized as follows.

Chapter 2 completely specifies the QC-MDPC McEliece encryption scheme; including key generation, encryption, and decryption, and discusses requirements of the decoding algorithm used in decryption. Chapter 3 includes the security reduction for the encryption scheme, as well as an analysis of the scheme's security against state-of-the-art (and generic) classical and quantum attacks, and proposes parameter sets accordingly. Chapters 4, and 5 accomplish the same as the second through third chapters, but for the KEM as opposed to the encryption scheme. Chapter 6 discusses the performance of the proposed KEM.

2. The QC-MDPC McEliece Encryption Scheme

2.1 Encryption Scheme Specification

This section fully describes the QC-MDPC McEliece encryption scheme. Each algorithm is described in a fairly generic way so as to facilitate their understanding. Possible optimizations and speedups are briefly discussed for each of the provided algorithms. For a more detailed description of possible optimizations, see the accompanying optimized implementation included with this submission.

2.1.1 Key Generation

To construct an (n, r, w)-QC-MDPC code is to construct its parity-check matrix. This document is only concerned with the case when $n = n_0 r$ where r is prime. In this case the parity-check matrix will have the form

$$H = [H_0|H_1|\dots|H_{n_0-1}],$$

where each H_i is itself a circulant $r \times r$ matrix. To construct such a parity-check matrix its first row need only be generated. This is done by randomly selecting a length n binary vector h of weight $w \in O\left(\sqrt{n \log(n)}\right)$ and parsing h into n_0 length r substrings as follows:

$$h = [(h_0, h_1, \dots, h_{n_0-1}), (h_{n_0}, h_{n_0+1}, \dots, h_{2n_0-1}), \dots, (h_{(r-1)n_0}, h_{(r-1)n_0+1}, \dots, h_{rn_0-1})].$$

Subvector $(h_{in_0}, \dots, h_{(i+1)n_0-1})$ is the first row of H_i - with the rest of H_i obtained by sequential cyclic shifts of its first row. In this way, each of these subvectors have their own weight w_i and $w = \sum_{i=0}^{n_0-1} w_i$.

By applying the fact that $HG^T = 0$ for a generator matrix G and its parity-check matrix H together with the assumption that H_{n_0-1} is invertible, one

can calculate a generator matrix in reduced row-echelon form as $G = [I_k \mid Q]$ where,

$$Q = \begin{bmatrix} (H_{n_0-1}^{-1} H_0)^T \\ (H_{n_0-1}^{-1} H_1)^T \\ \vdots \\ (H_{n_0-1}^{-1} H_{n_0-2})^T \end{bmatrix}$$

and I_k is the $k \times k$ identity matrix; recalling that $r = n - k$. If H_{n_0-1} is not invertible then it must be recalculated. However, in practice there is a high probability that it will be invertible.

There is a subtlety here that merits mentioning. The generator matrix G is not in general a generator matrix for a quasi-cyclic code, but rather it is isomorphic to such a generator matrix. However, as it turns out, the representation of G given above is suitable for the needs of the cryptosystem and furthermore, using this representation of G does not degrade security at all. The particular details are not important for this document and are thus omitted, but essentially to obtain a generator matrix G' for a QC-MDPC code from a matrix G as above, one must interleave the columns of G (a simple permutation).

Note that indeed G is a $k \times n$ matrix and that for any vector $x \in \mathbb{F}_2^k$, the first k bits of xG exactly equal x itself. We can now present the key generation algorithm for QC-MDPC McEliece.

Algorithm 1 QCMDPC.KeyGen

Input: Security parameter n, weight w, co-dimension r, and error-correction threshold t.

Output: Public key G, secret key H.

1: Generate a parity-check matrix $H \in \mathbb{F}_2^{r \times n}$ of a t-error-correcting (n, r, w)-QC-MDPC code as described above.
2: Calculate $G = [I_k \mid Q]$ as described above.
3: **return** (G, H).

The value t in the above depends on the decoding algorithm employed. See Section 2.1.4 for more details.

Key sizes: A QC-MDPC McEliece public key has size nk. However, the entirety of G need not necessarily be stored. As G always contains the $k \times k$ identity matrix, only the submatrix Q need be stored. The size of Q is $k(n-k) = k^2$. However, by using the fact that the block $(H_{n_0-1}^{-1} H_i)^T$ are themselves circulant matrices the storage requirements can be further reduced as one only need store the first rows. In total then, this requires

$(n_0 - 1)k$ bits of storage. In the case where $n_0 = 2$ (the case this document is most concerned with), only k bits need be stored. Similarly the secret key H is nk bits, but only the first row (consisting of n bits), or a secret seed used to generate the row need be stored.

Run time: In this generic way, computation of the parity-check matrix requires producing n bits of randomness, parsing said randomness into n_0 substrings, and performing $n_0 k$ subsequent cyclic shifts. As we discuss in Section [what section?], n_0 is typically taken to be 2, and so the value of k largely determines this cost. Moreover, the entire secret key need not be generated because, the first row provides enough information to efficiently generate the corresponding generator matrix. Computation of the generator matrix requires one $r \times r$ matrix inversion, and $n_0 - 1$ multiplications and transpositions of $r \times r$ matrices. If we make the *highly* conservative assumption that matrix inversion and each matrix multiplication takes r^3 operations, computation of G takes approximately $n_0 r^3$ operations (the cost of the transpositions can be safely excluded from this analysis). In actuality, an optimized implementation of this algorithm would run in time roughly linear in r.

2.1.2 Encryption

Encryption in the QC-MDPC McEliece scheme can be succinctly described as a matrix multiplication followed by an **xor** with an error vector. A generic description of this algorithm is described below.

Algorithm 2 QCMDPC.Encrypt

Input: Public key G, message $m \in \mathbb{F}_2^k$, and error vector $e \in \mathbb{F}_2^k$ of weight at most t.

Output: Ciphertext $c \in \mathbb{F}_2^n$.

1: $c \leftarrow mG \oplus e$.
2: **return** c.

Observe that this algorithm takes an error vector e as input. Most authors calculate the error vector within the encryption algorithm. However, this document does not propose QC-MDPC McEliece encryption for standardization, but rather an associated KEM. As such (and as the reader will see in Chapter 5) this variant of QC-MDPC encryption is desirable for the needs of this document.

Ciphertext size A ciphertext in this scheme is a compact n bits.

Run time The matrix/vector multiplication can be done very quickly; for example, recall that $G = [I_k \mid Q]$, and so the first k elements of this mG exactly equals m hence no computation is required to compute those bits. The **xor** operation takes trivial amounts of resources.

2.1.3 Decryption

Decryption requires as a subroutine a t-error-correcting QC-MDPC decoding algorithm with knowledge of the secret key H. Denote by this decoder $\quad_H$. For further details on this decoder, see Section 2.1.4.

Algorithm 3 QCMDPC.Decrypt

Input: Ciphertext $c \in \mathbb{F}_2^n$ and dimension k.
Output: Vector $m \in \mathbb{F}_2^k$ such that $d(mG, c) \leq t$, or $\perp$.

1: Compute $mG = \quad_H(c) = \quad_H(mG \oplus e)$. If this step fails output $\perp$.
2: Extract m as the first k bits of mG.
3: **return** m.

Run time The decryption algorithm takes time and resources essentially equal to that of the decoding algorithm.

2.1.4 Decoding

When it comes to decoding algorithms for (QC-)MDPC codes, one has a variety of options. There are two basic families of decoding algorithms: those of the Berlekamp et. al. variety [5] (note that McEliece himself was an author of that paper), and those of the Gallager variety [14]. For reasons discussed below, this submission employs a Gallager styled decoding algorithm; henceforth referred to as a *bit-flipping algorithm*.

The style of decoder put forth by Berlkamp et. al. does provide a lower decoding failure rate (more on failure rates later) which is desirable, but this advantage is countered by the fact that those styles of decoders are much more computationally complex and involve tedious floating-point arithmetic. On the other hand, bit-flipping algorithms are much more computationally simple.

Maurich et. al. in [28, Section 3.1] give a good high-level description of the guiding principles of bit-flipping algorithms. We reiterate this description below.

1. Compute the syndrome of the received ciphertext $s^T = Hc^T$.

2. Count the number of unsatisfied parity-check equations (c.f Definition 2.1.2) denoted $\#_{\text{upc}}$ associated with each ciphertext bit.
3. Flip each ciphertext bit that violates more than b equations (for some pre-determined positive integer b).
4. Recompute the syndrome of the updated ciphertext.
5. Repeat this process until one of the following events occur:
   (a) The syndrome computed equals 0, in which case the decoder is successful and outputs the corrected code.
   (b) A pre-defined maximum number of iterations is reached, in which case the decoder fails and outputs $\perp$.

To understand how and why a bit-flipping algorithm works, we find it necessary to first discuss it in the languages of Graph Theory and Linear Algebra. This is done below.

Definition 2.1.1 (Tanner Graph)**.** *Let H be a $k \times n$ parity-check matrix for a QC-MDPC code $\mathcal{C}$. Then the* Tanner graph *of H is the bipartite graph*[1] *(with partite sets A and B) obtained from H as follows.*

- *A contains one node for each row of H. These nodes are denoted as $f_0, f_1, \ldots, f_{k-1}$,*
- *B contains one node for each column of H. These nodes are denoted as $c_0, c_1, \ldots, c_{n-1}$.*
- *Vertex f_i is adjacent to (has an edge between) c_j if and only if the ij^{th} entry of H equals* 1*.*

The vertices of A are called check nodes*, and the vertices of B are called* variable nodes*.*

The Tanner graph is due to (as the name somewhat suggests) Michael Tanner in [47]. In what follows we denote the neighbours (vertices with whom an edge is shared) of vertex f_i by $c_{i_0}, c_{i_1}, \ldots, c_{i_{w-1}}$. Note that the notation c_j is also used to refer to the j^{th} coordinate of a ciphertext c (which we are presumably trying to decode); this notation is intentional. Moreover, as f_i corresponds to a row of H, which has weight w, it follows that each f_i indeed has w neighbours. The variable nodes can themselves be thought of as the current coordinates of the ciphertext. As their name suggests, these variables may *vary* (i.e., change) as the algorithm iterates. Below is an important definition which highlights this fact.

[1] That is its vertex set V set can be divided into disjoint sets A and B such that no two vertices in A (resp. B) have an edge joining them.

Definition 2.1.2 (Parity-check equation)**.** *Let f_i be a check vertex, and let variable node c_{i_j} be a neighbour of f_i. Then the* parity-check equation *for the pair (f_i, c_{i_j}) is the equation:*

$$\bigoplus_{\ell \neq j} c_{i_\ell} = c_{i_j}.$$

If equality fails to hold, then that parity-check equation is called unsatisfied *by c_{i_j}.*

More generally, we may refer simply to "the parity-check equation for f_i" as $\bigoplus_\ell c_{i_\ell} = 0$ if we do not need to specify a particular c_{i_j}. However, if we are asking if a particular c_{i_j} satisfies the equation, we will generally put that variable on the right hand side.

Below is a general framework for a graph-based bit-flipping algorithm. This framework assumes as input a parity-check matrix H, and a ciphertext c which it is trying to decode. This framework is based on that found in [25, Section 3.1].

1. Each variable node c_j sends to each of its neighbouring check nodes the value "it believes is the correct value" for bit c_j. The only information node c_j has at this step is the j^{th} bit of c. Thus, node c_j forwards this information to its neighbours.

2. In the second step, each check node computes and sends a response to each of its neighbours. The check nodes send to their neighbours the values they "believe" to be the correct values for them (the variable nodes). The check nodes have more information at this step than the variable nodes had in the previous step, and so this calculation is slightly less trivial. For each neighbour c_{i_j} of f_i, node f_i sends the value c_{i_j} would need to be in order to satisfy the (f_i, c_{i_j})-parity-check equation to node c_{i_j}.

3. In the third step, each variable node uses the data they have sent and received to determine which bits in the ciphertext have been corrupted. At this point, there are many different ways to proceed with bit flipping.

4. If all parity-check equations are satisfied by the current values of the c_j then the algorithm terminates and outputs the current ciphertext; else, return to step 2.

At a high-level, the algorithm essentially finds and flips the bits of c which are most likely to be corrupted.

The last two steps in the above offer some wiggle room so to speak. In particular, what methods can the variable nodes employ to decide if they

should be flipped or not? One way is by majority rule; the check node assigns to itself whatever value is most common amongst the data it has (guessing in the case of a tie). More often though, this decision is based on some pre-determined (but not necessarily fixed) threshold. For example, only flip bits who fail to satisfy "too many" parity-check equations. This second method is essentially the one proposed by Misoczki (et. al.) and Gallagher [34, 33, 14]. Moreover, one does not need to flip every bit at the same time. Some decoding algorithms only flip one bit at a time, then recomputes the syndrome, and then compares that updated syndrome to 0 before flipping the next bit (and terminates if equality holds).

Now we translate this graph theoretic approach into the language of Coding Theory. Observe that since a word is a codeword if and only if it is annihilated (mapped to 0) by the parity-check matrix, then if we recompute the syndrome after flipping some bits and the result is 0 it must be the case that the ciphertext was corrected to a valid codeword. Moreover, the weight of the error vector used in encryption is such that the decoding algorithm (if it outputs anything except $\perp$) outputs the correct, uncorrupted message by using Theorem 2.1.1 below (recalling Definition 1.1.5).

Theorem 2.1.1 ([26])**.** *A linear code with minimum distance d_0 can correct (decode) up to $\lfloor \frac{d_0-1}{2} \rfloor$ errors.*

Thus, a necessary condition for a ciphertext to be properly decoded is that the weight t of the error vector is at most $\lfloor \frac{d_0-1}{2} \rfloor$. The intuition here is that so long as not more than $\lfloor \frac{d_0-1}{2} \rfloor$ bits are flipped during the encoding process then the correct vector is still the closest codeword. Moreover, if the computed syndrome is 0, then we are guaranteed that we have found that correct vector.

In general, calculating the minimum distance of a binary code is NP-hard [48]. Hence, one runs into a problem when performing parameter selection, namely, how does one select a secure value $t \leq \lfloor \frac{d_0-1}{2} \rfloor$? It turns out that there are some reliable heuristic techniques one can employ for this purpose. One option is to work from Gallager's own analysis from [14] to establish an upper bound on the error correction capability of the code. Alternatively, it is possible to estimate this value instead in terms of the decoding failure rate (DFR) of the code; which can itself be estimated reliably by running the algorithm many times.

How does the concept of a parity-check equation translate in terms of matrix multiplication? Recall that f_i and c_j are adjacent if and only if the j^{th} entry in the i^{th} row of H is 1. Hence, the parity check equation $\bigoplus_\ell c_{i_\ell} = 0$ for check node f_i can equivalently be called the parity-check equation for row i, and expressed as $R_i c^T = 0$, where R_i is the i^{th} row of H. Hence, all

parity-check equations are embedded in the equation $Hc^T = 0$. In other words, non-zero coordinates in the syndrome of the ciphertext correspond to unsatisfied parity-check equations.

This document does not propose any particular decoding algorithm for standardization. This is because a standardized decoding algorithm is not required for interoperability and is hence left as a decision to be made on an implementation basis.

3. Security of the QC-MDPC McEliece Encryption Scheme

This chapter gives a rather complete analysis of the security of the QC-MDPC McEliece encryption scheme. We begin the chapter by presenting and discussing the theoretical proof of security for the scheme in Section 3.1. In Section 3.2 we move on to the more concrete security considerations of the scheme; including in-depth analyses of the various known attacks against the encryption scheme. We conclude this chapter in Section 3.3 by giving a collection of suggested parameter sets, as well as by giving a simple and general method for computing parameter sets providing s-bits of security.

3.1 Theoretical Security Analysis

Misoczki et. al. in [34] give a theoretical proof of security for the QC-MDPC McEliece encryption scheme. We restate the result here, but refer the reader to the source material for a more complete proof.

First we give some notation, then define three "hard problems".

- $\mathcal{F}_{n,r,w}$ is a t-error correcting family of (n, r, w)-QC-MDPC codes.
- $\mathcal{K}_{n,r,w}$ is the key space of $\mathcal{F}_{n,r,w}$.
- $\mathcal{H}_{n,r}$ is the set of all full-rank, circulant block matrices in $\mathbb{F}_2^{r\times n}$. Note that necessarily $\mathcal{K}_{n,r,w} \subsetneq \mathcal{H}_{n,r}$.
- $\mathcal{S}_n(0, t)$ is the n-dimensional sphere centered around 0 with radius t.

The Code Distinguishing Problem: Given parameters $\mathcal{H}_{n,r}, \mathcal{K}_{n,r,w}$, and a problem instance $H \in \mathcal{H}_{n,r}$, decide if $H \in \mathcal{K}_{n,r,w}$.

Codeword Existence Problem: Given parameters $\mathcal{H}_{n,r}$ and $w \in \mathbb{Z}^+$ and a problem instance $H \in \mathcal{H}_{n,r}$, decide if there exists a codeword of weight at

most w in the code generated by H.

The Computational Syndrome Decoding Problem: Given parameters $\mathcal{H}_{n,r}$ and $t \in \mathbb{Z}^+$ and a problem instance $(H, s) \in \mathcal{H}_{n,r} \times \mathbb{F}_2^r$, produce a vector $e \in \mathcal{S}_n(0, t)$ such that $eH^T = s$.

We state the following result from [34, Section 5].

Proposition 3.1.1. *For parameters $\mathcal{H}_{n,r}$, and $\mathcal{K}_{n,r,w}$ and assuming that solving the code distinguishing problem for these parameter is not easier than solving the codeword existence problem for parameters $\mathcal{H}_{n,r}$ and w then breaking the QC-MDPC McEliece encryption scheme is not easier than solving the computational syndrome decoding problem for a random quasi-cyclic linear code.*

3.2 Analysis Against Known Attacks

This section describes and discusses the known attacks against the QC-MDPC McEliece encryption scheme. This of course includes a variety of classical attacks, quantum attacks, and side-channel attacks. Included in this section is discussions on the complexity and efficacy of these attacks as well as methods to prevent the attacks or reduce their effects. We begin this crucial discussion by giving a quick review of Grover's algorithm [18] and its generalized variant [7].

3.2.1 Grover's Algorithm

There are several possible interpretations of the problem which Grover's algorithm solves. The most general one may be computing a black-box Boolean function; meaning an unstructured search. This is equivalent to computing the root of a particular function if such a root exists. More formally, suppose one has an oracle $f\colon \{0,1\}^n \to \{0,1\}^m$ (also called a black-box function) and an image y^* with the promise that there exists *one* input x such that $f(x) = y^*$. The problem is to find the *solution* x. We can reformulate the problem using a predicate $P(x)$:

$$P(x) = \begin{cases} 1 & \text{if } f(x) = y^* \\ 0 & \text{otherwise} \end{cases}$$

Let $N = 2^n$ be the size of the search space or domain. Classically, there is no better strategy to find this x than trying distinct inputs at random until an x is found such that $f(x) = y^*$. This requires trying $N/2$ inputs on average

and $N-1$ in the worst case. However, Grover's algorithm [18] optimally solves this problem in

$$O(\sqrt{N})$$

quantum queries to f with bounded-error probability $O(1/N)$. A query refers to the evaluation of f on some input. Therefore, Grover's algorithm is provably more efficient than any classical algorithm for search problems modelled as a black-box.

When the problem has s solutions, then a generalized version of Grover's algorithm can find a solution in

$$O(\sqrt{N/s})$$

queries to the black-box function. This variant is essential in this work. Again, this is optimal [7]. Readers interested in more details are referred to [23, 41, 21].

Theorem 3.2.1 (Quadratic speed-up with known p [7])**.** *Let $\mathcal{A}$ be any quantum algorithm that uses no measurements, and let $f : Z \to \{0,1\}$ be any Boolean function. Let $p > 0$ denote the initial success probability of $\mathcal{A}$. Then there exists a quantum algorithm that finds a solution with certainty using a number of applications of $\mathcal{A}$ and $\mathcal{A}^{-1}$ which is in $\Theta(1/\sqrt{p})$.*

3.2.2 Information Set Decoding

There are two main approaches for attacking code-based cryptosystems: (1) private-key recovery; and (2) message recovery from the ciphertext without the private key. The best algorithm currently known for all these attacks is Information Set Decoding (ISD), which was originally proposed by Prange [39]. Next, we explain in detail how this algorithm works by applying it to QC-MDPC McEliece.

Consider an (n,k,t)-linear code, and let $c = mG+e$ be a given n-bit ciphertext. Let $I = \{i_1, i_2, \ldots, i_k\}$ be a k-subset of $\{1,2,...,n\}$. Let $v_I = v_{i_1} v_{i_2} \ldots v_{i_k}$ be the vector consisting of the k components of a vector $v \in \mathbb{F}_2^n$ indexed by I, and $\pi_1 \colon \mathbb{F}_2^n \to \mathbb{F}_2^k$ the projection that performs this operation. Let $G_I = G_{i_1} G_{i_2} \ldots G_{i_k}$ be the submatrix consisting of the k columns $i_1, i_2, \ldots, i_k$ of G. Finally, π_2 denotes the projections that sends G to G_I. Since $c = mG+e$

is equivalent to the system of equations

$$\begin{aligned} c_1 &= mG_1 + e_1 = m_1 g_{11} + m_2 g_{21} + \cdots + m_k g_{k1} + e_1 \\ c_2 &= mG_2 + e_2 = m_1 g_{12} + m_2 g_{22} + \cdots + m_k g_{k2} + e_2 \\ &\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\ c_n &= mG_n + e_n = m_1 g_{1n} + m_2 g_{2n} + \cdots + m_k g_{kn} + e_n \end{aligned}$$

we can write

$$c_I = mG_I + e_I.$$

The crucial observation is that if all the k components of e_I are zeros and G_I is invertible, then $c_I = mG_I$ and one can conditionally recover the plaintext using:

$$m = c_I G_I^{-1}.$$

At this level we have all the material to explain the ISD algorithm. The idea of ISD is to repeatedly select k bits at random from c to form a k-bit vector c_I where hopefully none of the selected bits has an error. More precisely, the adversary runs the following function (*ISD Algorithm*):

1. Select a random k-subset $I = \{i_1, i_2, \ldots, i_k\} \subset \{1, 2, ..., n\}$;
2. Choose k bits from c according to I to form $c_I = c_{i_1} c_{i_2} \ldots c_{i_k} = \pi_1(c)$;
3. Choose k columns from G according to I to form a matrix $G_I = \pi_2(G)$;
4. Compute G_I^{-1} the inverse of G_I if it exists, otherwise Go to Step 1;
5. Compute $m = c_I G_I^{-1}$;
6. Compute $e = c + mG$;
7. If the weight of e is greater than t, then Go to Step 1;
8. Return m as the correct plaintext.

The required computational work is calculated as follows [30, 40, 2]. The error vector consists of t ones and $n - t$ zeros. The probability of choosing k zeros from e is $\binom{n-t}{k}/\binom{n}{k}$. The probability of choosing a uniformly random matrix from the set of $k \times k$ binary matrices is about 0.288 [8], and may be smaller in the case of G_I. To be conservative, assume it is about 1/2. Actually, this little advantage will disappear in the quantum case. In this case, the total success probability is $p \approx \binom{n-t}{k}/2\binom{n}{k}$. On average, an adversary $\mathcal{A}$ has to make $1/p$ attempts (iterations) to recover the plaintext, and for each iteration only a few $k \times k$ matrix operations are required. Assuming matrix

inversion takes k^γ operations, for $2 < \gamma < 3$, the total work factor of the (classical) basic ISD is

$$\mathrm{WF}_{\mathrm{isd}} \approx k^\gamma/p \approx 2k^\gamma \binom{n}{k} / \binom{n-t}{k}.$$

An approximation formula shows that the total work factor can be written as

$$\mathrm{WF}_{\mathrm{isd}} \approx 2^{(\alpha(R,W)+o(1))n}$$

where $\alpha(R,W)$ is a positive function, and $R = k/n$ and $W = t/n$ are positive constants that represent the code rate and error fraction of the code, respectively. This upper bound arises from an asymptotic binomial coefficient optimization:

$$\log \binom{n}{k} = (1+o(1))nH\left(\frac{k}{n}\right),$$

where $H(p) = -p\log p - (1-p)\log(1-p)$ is the binary entropy. See [37, 4, 20] for more details on this upper bound.

The following formula allows to compute α to high precision for any (R,W):

$$\alpha(R,W) = (1-R-W)\log(1-R-W) - (1-R)\log(1-R) - (1-W)\log(1-W).$$

There have been many improvements [24, 46, 10, 29, 4] over the basic ISD [39]. However, all efforts have only managed to slightly decrease the exponent $\alpha(R,W)$ in such a way that the asymptotic cost is still exponential in $\alpha(R,W)$. Note that these variants are faster than the original ISD, and used to assess the classical security. However, it's difficult to design their quantum counterparts, if there are any. One of the reasons for this is memory constraints [20]. It turned out that Grover's algorithm combined with the basic ISD is sufficient to assess the quantum security of code-based cryptosystems.

Remark 3.2.1. *There are a few methods to verify whether the value $m = c_I G_I^{-1}$ is indeed the true plaintext. One method requires a lot of redundancy in the plaintext [40]. A more practical solution is the "systematic method" used in Step 7, which was introduced in [24]. It works because G generates a code of minimum distance larger than $2t$.*

3.2.3 Quantum Information Set Decoding

The QISD algorithm is a quantum version of the ISD algorithm. There are several conceivable QISD algorithms, depending on the ISD variant and also on the quantum search tool (Grover's algorithm, quantum walks [43]). Here we consider the original ISD along with Grover's algorithm [6]. The other

variants essentially provide the same asymptotic complexity. They all reduce the classical work factor from $2^{(\alpha(R,W)+o(1))n}$ to $2^{(\alpha(R,W)+o(1))n/2}$. Before presenting QISD, we need to define the crucial classical function that should be converted into a quantum oracle for Grover's algorithm.

Given a k-subset $I \subset \{1, 2, ..., n\}$ and possibly other inputs, consider the following *boolean* function f assumed to be available as a *classical oracle*:

1. Compute c_I by applying on c the projection $\pi_1 \colon \mathbb{F}_2^n \to \mathbb{F}_2^k$;
2. Compute G_I by applying on G the projection $\pi_2 \colon \mathbb{F}_2^{k \times n} \to \mathbb{F}_2^{k \times k}$;
3. Compute the inverse G_I^{-1} if it is invertible, otherwise return 0;
4. Compute $m = c_I {G_I}^{-1}$;
5. Compute $e = c + mG$;
6. If the Hamming weight of e is greater than t, then return 0;
7. Return 1 (True).

More concisely, the function f can be written as:

$$f(I) = \begin{cases} 1 & \text{if } G_I \text{ is invertible and weight of } e \leq t \\ 0 & \text{otherwise} \end{cases}$$

Let U_f be the quantum unitary, also called quantum oracle, that implements the function f in the standard way:

$$U_f \colon |x, b\rangle \to |x, b \oplus f(x)\rangle$$

for all input x and Boolean b. This convenient notation in the quantum field simply says that (x, b) is mapped to $(x, b \oplus f(x))$. The input is kept to make the circuit reversible, which is a fundamental requirement in quantum computing. Note that if f is efficient, then U_f is efficient (See Remark 3.2.2).

Remark 3.2.2. *All calculations performed efficiently on classical computers can also be performed efficiently on quantum computers [21]. One essentially has to replace classical gates with reversible gates and bits with qubits, adding extra qubits needed for reversibility.*

A quantum algorithm needs at least two quantum registers (collection of qubits), an *index* (or subset) register R_I and an *answer* or *oracle* register R_q, neglecting the working register for simplicity. The QISD algorithm have the following steps:

1. Prepare a register R_I in superposition of all possible k-subsets of integers I, that is, $I = \{i_1, i_2, \ldots, i_k\} \subset \{1, 2, ..., n\}$;

2. Compute f for all possible inputs I by applying U_f on the register R_I; the corresponding oracle values will appear in the entangled register R_q;
3. Perform Grover's iteration (efficient quantum transformations);
4. Repeat steps 1 through 3 approximately $\frac{\pi}{4}\sqrt{1/p}$ times;
5. Read (measure) the result, and if the value of R_q is not 1, then restart.

The analysis of this algorithm is exactly the same as ISD except that the required number of iterations is approximately $\frac{\pi}{4}\sqrt{1/p}$ thanks to Grover's algorithm, or more generally, amplitude amplification (see Theorem 3.2.1). Each iteration consists of evaluating the quantum oracle and takes time in the order of k^γ qubit operations, with $\gamma < 3$, since it is dominated by matrix operations. Thus, the total quantum work factor is

$$\mathrm{QWF}_{\mathrm{qisd}} \approx k^\gamma/\sqrt{p} \approx 2^{(\alpha(R,W)+o(1))n/2}. \tag{3.1}$$

Accordingly, there is a quadratic speed-up over known classical attacks. However, this doesn't necessarily translate into doubling the public key size in order to maintain the security level.

Application: Consider for instance McEliece using Goppa codes where the public key is a $k \times n$ matrix. Protecting against this attack requires replacing n by $(2 + o(1))n$. This essentially quadruples the McEliece key size because $k \geq n - t\log(n)$, thus we have to double both k and n. For QC-MDPC McEliece, there is no need to quadruple the public key size.

3.2.4 Asymptotic Quantum Security

The goal of this section is to analyze the quantum security of the QC-MDPC McEliece cryptosystem using the original ISD combined with generalized Grover's algorithm in a slightly more general way than in [6] because of the additional structure of this scheme. We first start with a summary of the classical security from [34].

Consider the system as an instantiation of the McEliece (or Niederreiter) scheme using an (n, r, w)-QC-MDPC code with capability of correcting t errors. The claim is that the best attacks are:

- *Key distinguishing attack*: distinguish the public key from a random matrix, which invalidates the security reduction.
- *Key-recovery attack*: Recover the secret decoder (private key)

- *Decoding attack*: recover the plaintext from the ciphertext without using the private key, i.e. decode t errors in (n, r)-linear code.

ISD work factor is the standard approach to estimate the practical security of code-based schemes. However, the cryptanalysis of QC-MDPC McEliece is subtle. The problem of finding a single low weight codeword in an MDPC code may admit many solutions, making ISD faster. The DOOM attack [44] in the classical setting can decrease the work factor when the problem has multiple solutions and the attack is considered successful when a single solution is found. When the problem has N solutions, the probability of finding a solution increases by a factor of N. In the quantum case, this probability increases by a factor of $\sqrt{N}$. Accordingly, we have to use Grover's algorithm considering many solutions to the problem in order to provide an accurate security assessment.

Let $\mathrm{WF}_{\mathrm{isd}}(n, r, t)$ denote the classical work factor of decoding t errors in an (n, r)-binary linear code of length n and co-dimension r, when there is a single solution to the problem. It is also the cost of finding a codeword of weight t. Let $\mathrm{WF}'_{\mathrm{isd}}(n, t, t)$ be the cost of $\mathrm{WF}_{\mathrm{isd}}(n, r, t)$ without oracle calls; it is the classical number of iterations. Let k^γ be the cost of each quantum oracle call with $\gamma < 3$. The following shows the quantum work factors of the attacks mentioned at the beginning of this section, currently considered to be the most efficient attacks on QC-MDPC McEliece.

1. Key distinguishing attack: Produce one word of weight w in the dual code by applying ISD to the all-zero syndrome. This problem has r solutions (the r rows of the sparse parity-check matrix). Assuming there is "no obvious speed-up" in the quasi-cyclic case [34], the work factor of this attack in the quasi-cyclic and non-cyclic cases are the same. Therefore, using Grover's algorithm with r solutions, the quantum work factor of key distinguishing attack is

$$\mathrm{QWF}^{\mathrm{QC}}_{\mathrm{dist}}(n, r, w) \approx \mathrm{QWF}_{\mathrm{dist}}(n, r, w) \approx k^\gamma \sqrt{\frac{\mathrm{WF}'_{\mathrm{isd}}(n, n-r, w)}{r}}.$$

2. Key recovery attack: Finding one codeword in the basis of the dual code is sufficient to recover the entire private key, and there are r such codewords. Therefore, applying Grover's algorithm with r solutions, the quantum work factor of this attack in the quasi-cyclic case is the same as $\mathrm{QWF}^{\mathrm{QC}}_{\mathrm{dist}}(n, r, w)$:

$$\mathrm{QWF}^{\mathrm{QC}}_{\mathrm{reco}}(n, r, w) \approx k^\gamma \sqrt{\frac{\mathrm{WF}'_{\mathrm{isd}}(n, n-r, w)}{r}}.$$

3. Decoding attack: Correct t errors in a random linear code of length n with co-dimension r. The classical work factor of this attack is

$$\mathrm{WF}_{\mathrm{dec}}(n, r, t) = \mathrm{WF}_{\mathrm{isd}}(n, r, t).$$

When the code is quasi-cyclic, any cyclic shift of the target syndrome provides a new instance whose solution is equal to the one of the original syndrome, up to a block-wise cyclic shift. The number of solutions (and instances) is r. However, Grover's algorithm is only running on one instance (not parallel). Therefore, the quantum work factor of decoding attack in the quasi-cyclic case is

$$\mathrm{QWF}_{\mathrm{dec}}^{\mathrm{QC}}(n, r, t) \approx k^{\gamma}\sqrt{\mathrm{WF}_{\mathrm{isd}}(n, r, t)}$$

Typically, the work factor of decoding attack is different from that of key recovery attack. Therefore, the least work factor determines the security level. See more numeric details in the following section.

3.2.5 Practical Quantum Security

The formulae in the previous section provide a good measure about the asymptotic quantum security of QC-MDPC. In practice, concrete parameters are needed.

Given parameter sets for QC-MDPC McEliece classical security, this section shows their quantum security by applying the general formulae given previously. First of all, we emphasize the approach to achieve our goal.

Recall that the total quantum work factor is approximately $2^{(\alpha(R,W)+o(1))n/2}$ (see eq. (3.1)). First, we compute the required number of iterations, which is about $2^{(\alpha(R,W))n/2}$. Second, we approximate the $o(1)$ factor essentially based on matrix operations. Our approach is fairly conservative. For instance, we consider a potential speed-up in the QISD algorithm and do not consider the cost of a fault-tolerant quantum implementation of the oracle, which increases the work factor of the attack [3, 17].

Now we give an example of evaluating the security using existing parameters. Table 3.1 shows parameter sets selected from [34]. Note that we only consider the case $n_0 = 2$ for reasons to be discussed in Section 3.3.1. Consider the parameter set $(n, r, w, t) = (65542, 32771, 274, 264)$ which provides approximately 256-bits of classical security. We compute the corresponding quantum security level of this parameter set.

The key distinguish/recovery attack is slightly more efficient than the decoding attack in this case. More precisely, given the number of solutions $k = r \approx 2^{16}$,

Classical security	n_0	$n = n_0 r$	r	w	t	Key size
	2	9602	4801	90	84	4801
80	3	10779	3593	153	53	7186
	4	12316	3079	220	42	9237
	2	19714	9857	142	134	9857
128	3	22299	7433	243	85	14866
	4	27212	6803	340	68	20409
	2	65542	32771	274	264	32771
256	3	67593	22531	465	167	45062
	4	81932	20483	644	137	61449

Table 3.1: Suggested parameter sets for classical security.

considering the quadratic speed-up from QISD, and adding the contribution of the $o(1)$ factor, the total quantum work factor is

$$\mathrm{QWF}_{\mathrm{reco}}^{\mathrm{QC}}(n, r, w) \approx 2^{154}.$$

We conclude that this parameter set provides at least 154-bit quantum security. Table 3.2 shows the results of similar calculations for the other security levels.

Quantum security	n_0	$n = n_0 r$	r	w	t	Key size
58	2	9602	4801	90	84	4801
86	2	19714	9857	142	134	9857
154	2	65542	32771	274	264	32771

Table 3.2: Suggested parameter sets for quantum security.

Remark 3.2.3. *The mitigation of the quadratic advantage due to quantum algorithms is not by simply doubling the public key size. This depends on the parameter set of the actual security level. More precisely, maintaining 128-bit quantum security requires tripling the public key size if we keep the same code rate (9857 vs 32771 bits).*

3.2.6 Other quantum attacks

The first attempt of using quantum algorithms to speed-up ISD appeared in [35]. The method of applying Grover's algorithm did not give significant speed-up over classical ISD algorithms. Bernstein [6] then showed that it is possible to obtain much better speed-ups with Grover's and Prange's

algorithms, which brings down the exponent by a factor of two. Using quantum walks and the MMT algorithm [29], the authors of [20] slightly decreased this exponent. However, their approach did not yield any new significant improvements. See Table 3.3.

Author	Strategy	Alpha
Bernstein [6]	Prange + Grover's algorithm	0.06035
KacTil17 [20]	MMT + Quantum walks	0.05869

Table 3.3: Summary of the-state-of-the-art quantum attacks

Although they essentially have the same cost, there are several reasons for which we preferred ISD with Grover's algorithm over those using quantum walks. We mention (1) Ease of implementation from the engineering point of view. The second approach requires (classical) memory that has to be accessed in superposition, making it much harder to implement; (2) Ease of adaptation and analysis.

3.3 Parameter Selection

In this section, we discuss the speed/key size trade-off involved in selecting a value for n_0. We also give a general method for computing parameter sets for a desired level of security.

3.3.1 Deciding the value of n_0

General formulae are necessary to compute adequate parameter sets. However, in the case of QC-MDPC McEliece, other information is still important in order to compute parameter sets more accurately. Indeed, for the same security level, there are many parameter sets depending on the code rate, or equivalently n_0 (see Table 3.1 showing classical parameter sets). This section discusses strategies to choose optimal trade-off parameters in general, and the reason for which we recommend using $n_0 = 2$ (or $R = 1/2$). First the work factor of decoding attack is less than that of the other attacks, and it is in turn much less in the cases of $n_0 = 3, 4$. This is what we call *unbalanced work factors of attacks*, and is mainly due to the difference between (or ratio of) w and t. Second the public key size is much larger, which harms the main achievement of the scheme. For $n_0 = 4$ for instance, the public key size becomes 61449 bits instead of 32771 bits. The advantage of choosing $n_0 = 3, 4$ is that the code rate is better, and key generation, encryption, and decryption

can be faster. Considering that security and public key size are the main concerns in this scheme, this document recommends using $n_0 = 2$.

3.3.2 Suggested Parameter Sets and Expected Security

The following table is essentially a combination of those from Section 3.2.5. For each suggested parameter set, we describe the security one should expect that set to provide. For reasons discussed in Section 3.3.1 only parameter sets using $n_0 = 2$ are detailed here. We reiterate that this document does not propose QC-MDPC McEliece encryption itself for NIST standardization, but rather an associated KEM. Thus, the following discussion is included for completeness.

Security							
Classical	Quantum	n_0	$n = n_0 r$	r	w	t	Key size
80	58	2	9602	4801	90	84	4801
128	86	2	19714	9857	142	134	9857
256	154	2	65542	32771	274	264	32771

Table 3.4: Suggested parameter sets for classical and quantum security.

We can pair up these parameter sets rather naturally by tuples of the form (C, Q), where C is the number of classical bits of security provided, and Q is the number of quantum bits of security provided. Using this notation, parameter sets $(80, 58)$ and $(128, 86)$ both offer less security than the weakest security strength category defined by NIST [1, Section 4.A]. As such, this document proposes neither of them for general usage. The $(256, 154)$ parameter set meets the requirements for level 3 security.

3.3.3 Computing other parameter sets

Based on [33, Section 6.6], the following iterative procedure allows for the computation of parameter sets for s bits of quantum security. The inputs are the code rate R, the security parameter s, and a decoding failure threshold.

Algorithm 4 Computing Parameter Sets

Input: Code length n, dimension k, and desired security level s.
Output: Parameter set (n, k, w, t).

1: Compute the minimum t such that $\mathrm{QWF}^{\mathrm{QC}}_{\mathrm{dec}}(n, r, t) > 2^s$.
2: Compute the minimum w such that $\mathrm{QWF}^{\mathrm{QC}}_{\mathrm{reco}}(n, r, w) > 2^s$.
3: **if** messages with t errors can be decoded with the decoding failure threshold **then**
4: **return** (n, k, w, t).
5: **else**
6: **return** $\perp$.
7: **end if**

4. The QC-MDPC McEliece KEM

4.1 KEM Specification

This section fully describes the QC-MDPC McEliece Key Encapsulation Mechanism. Each algorithm is described in a fairly generic way so as to facilitate their understanding. Optimizations and speedups are briefly discussed for each of the provided algorithms. For a more complete description of possible optimizations, see the accompanying optimized implementation included in this submission.

4.1.1 Key Generation

QC-MDPC KEM key generation is exactly the same as QC-MDPC key generation (Algorithm 1).

4.1.2 Encapsulation

Key encapsulation and decapsulation requires the use of a deterministic error vector derivation function which we identify with the notation $\nu(\cdot)$. Such a function is required to have the following three characteristics.

- The function must be pseudorandom,
- The function must be one-way, and
- The function must be "reasonably fast".

This document suggests the following construction for $\nu(\cdot)$, but in general any function that generates a uniformly random weight w bit array will suffice. Denote by b^ℓ the string consisting of ℓ copies of bit b.

In Algorithm 5 above, the subroutine $\mathsf{FY}(\textsc{seed}, \textsc{string})$ is the "inside-out"

Algorithm 5 Error vector derivation

Input: A random seed $s \in \mathbb{F}_2^k$, and a weight $t \in \mathbb{N}$.
Output: An error vector e of length n and weight t.

1: Set $e \leftarrow 1^t \| 0^{n-t}$.
2: Shuffle e as, $e \leftarrow \mathsf{FY}(s, e)$.
3: **return** e.

variant of the Durstenfeld implementation[11] of the Fisher-Yates Shuffle [13].

In addition to the above, the KEM also requires the use of two key-derivation algorithms: $\mathsf{KDF}_1 : \{0,1\}^* \rightarrow \{0,1\}^k$, and $\mathsf{KDF}_2 : \{0,1\}^* \rightarrow \{0,1\}^{m+n}$, where m is the length of the key to be encapsulated.

Next, we specify the encapsulation algorithm as Algorithm 6 below.

Algorithm 6 QCMDPC.Encap

Input: Public key G, and random seed $s \in \mathbb{F}_2^k$.
Output: Symmetric key $K \in \{0,1\}^m$
Output: Ciphertext $C = (C_1, C_2) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$.

1: $e \leftarrow \nu(s)$ ▷ Compute n-bit error vector
2: $y \leftarrow \mathsf{KDF}_1(e)$ ▷ Compute k-bit masking value
3: $x \leftarrow s \oplus y$ ▷ Obtain k-bit plain text
4: $C_1 \leftarrow \mathsf{QCMDPC.Encrypt}(G, x, e)$
5: $C_2 || K \leftarrow \mathsf{KDF}_2(s)$
6: **return** $(K,\ C = C_1 || C_2)$

4.1.3 Decapsulation

The decapsulation algorithm is presented as Algorithm 7 below.

Algorithm 7 QCMDPC.Decap

Input: Secret key H, ciphertext $(C_1, C_2) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, and dimension k.
Output: Symmetric key $K \in \{0,1\}^m$ or a decapsulation failure $\perp$.

$((x, e), d_{\text{err}}) \leftarrow \mathsf{QCMDPC.Decrypt}(H, C_1)$.
$y \leftarrow \mathsf{KDF}_1(e)$ ▷ Compute k-bit masking value.
$s \leftarrow x \oplus y$ ▷ Recover seed.
$e' \leftarrow \nu(s)$. ▷ Derive Error Vector
$C_2'||K \leftarrow \mathsf{KDF}_2(s)$. ▷ Derive Key and confirmation hash.
if $e' = e$ **and** $C_2' = C_2$ **and** $d_{\text{err}} =$ False **then**
    return K
else
    return $\perp$
end if

4.2 Ephemeral use of the QC-MDPC McEliece KEM

To completely negate the so-called GJS attack [19], the QC-MDPC KEM can be used ephemerally. Consequently, the protocol becomes 2-pass; between the initiator and responder. To establish a shared ephemeral key the initiator must first generate a QC-MDPC key pair and send the public-key G to the responder. The responder then selects a random seed $s \in \mathbb{F}_2^k$ and runs $\mathsf{QCMDPC.Encap}(s, G)$. Next, the responder sends (C_1, C_2) to the initiator. Finally, the initiator runs $\mathsf{QCMDPC.Decap}(C_1, C_2)$ to recover the encapsulated key. If all algorithms run successfully, then the initiator and responder will share a secret key K (which they might use as a seed to derive further keys). If any algorithms fail, then the process is terminated and restarted. This process is summarized in Figure 4.1 below.

Initiator		Responder
$(G, H) \leftarrow \mathsf{KeyGen}(n, r, w, t)$		
	$\xrightarrow{G}$	
		$(C_1, C_2, K) \leftarrow \mathsf{Encap}(s, G)$
	$\xleftarrow{(C_1, C_2)}$	
$K/\perp := \mathsf{Decap}(C_1, C_2)$		

Figure 4.1: Ephemeral Key Establishment

The above protocol is an unauthenticated key establishment protocol. This protocol can easily be modified into either a one-way or a two-way authenti-

cated key establishment protocol by following the SIGMA protocol design [22]. However, those details are outside the scope of this document.

4.3 Static Use of the QC-MDPC McEliece KEM

In some cases, an individual may wish to use the QC-MDPC KEM in a static key setting. This is useful because, it allows one-pass key transport with the KEM. However, extreme caution must be taken when using the KEM with a static generator matrix, as it opens up a new class of attacks, and in particular, the GJS attack [19] (see Section 4.3.1).

The GJS attack makes use of the fact that decoding failures occasionally occur in the QC-MDPC decryption process. While the attack requires a large number of decoding failures to have happened to recover the parity check matrix, it is not known if the attack can be improved greatly more, or if another attack is possible if even a small number of decoding failures occur.

However, as long as a decoding error does not occur, the scheme is secure in a static-key context. Therefore the scheme can be thought to be resistant to accidental key reuse. In the event that a generator matrix and parity check matrix are reused for multiple sessions, this does not result in a security vulnerability as long as a decoding error does not occur in further sessions.

4.3.1 A Key Recovery Attack

On occasion the decoding algorithm fails to recover the error vector used in encryption. This is known as a *decoding error*. For our main parameter set, with uniformly random errors, decoding errors occur at a rate of roughly 10^{-7}.

The GJS attack [19] is a *key recovery attack* based on the observation that there exists some correlation between the error vectors used in encryption that result in decoding failures, and the secret key H. Specifically, when the distances between pairs of 1's in the error vector match the distances between pairs of 1's in the secret key, a decoding failure is less likely to occur; these are simply referred to as *distances* below. It is important to note that the plaintext x never has an impact on whether a decoding error occurs; only the error vector does.

The set-up for the GJS attack is roughly summarized as Algorithm 8 below.

For each possible distance in an error vector, the algorithm counts how many times that distance was present in error vectors that resulted in a decoding error.

Algorithm 8 GJS

Input: $M \in \mathbb{N}$.
Output: d_o : Frequency array of distances observed.
Output: d_e : Frequency array of errors when distance was observed.

1: **for** $i = 1, 2 \ldots, M$ **do**
2: $(mG, e) \leftarrow$ Random Encryption.
3: Send $mG \oplus e$ to the target.
4: Receive Success or Decoding Error from target.
5: **for all** $d \in$ distance-spectrum(e) **do**
6: $d_o[e] \leftarrow d_o[e] + 1$
7: **if** Decoding error occurred **then**
8: $d_e[e] \leftarrow d_e[e] + 1$
9: **end if**
10: **end for**
11: **end for**
12: **return** (d_o, d_e)

After a large number of submitted ciphertexts, the adversary can calculate what the decoding failure rate was when a particular distance was present in the error vector. The authors of [19] noted that when the decoding failure rate for when a given distance is in the error is calculated this way, it follows an approximately normal distribution with mean $m_{mult(d)}$. Here, $mult(d)$ (the multiplicity of d) is the number of times d is observed in the secret key. $m_{mult(d)}$ is a mean that is the same for all distances with the same multiplicity.

This allows one to figure out how often a distance appears between all the pairs of 1's in a secret key. In [19] the authors additionally show how a secret key can be computed by knowing how often each distance appears between the 1's in that secret key.

The attack requires that for each possible distance one can make an accurate guess about which mean $m_{mult(d)}$ the decoding failure rate should be associated with. As mentioned above, the attack allows one to make a sample of the decoding failure rate according to a normal distribution. Therefore the effectiveness of the attack depends on two crucial properties:

1. How many samples M are made.

2. The distance between the means $m_{mult(d)}$ for different multiplicities.
3. How quickly the variance of the decoding failure rate decreases with M.

Experimental evidence suggests for the 256-bit parameter set, the normal distribution can be modelled as a limit of a binomial distribution. With this assumption, one can calculate how the variance changes with M, and in particular, how large M should be in order to have the standard deviation small enough to be able to accurately tell the multiplicity of a distance in the secret key from this sample. Initial experiments and estimates indicate that for the 256-bit parameter set, this corresponds to around 3.5 trillion ciphertext queries.

While allowing for 3.5 trillion cipher text operations may be more than suitable for many purposes (such as S/MIME) it should be noted that this estimate only considers the effectiveness of the attack with no improvements whatsoever. In particular, even small improvements such as generating slightly non-uniform error vectors (through hash sampling) could dramatically reduce the attack requirements. Therefore, while it is not known if the parity check matrix of the KEM can be recovered when the key is reused thousands or even millions times, it should be considered highly insecure, and is not recommended for more general static key establishment.

4.3.2 Constant Time Decoders

In [19], the attack took advantage of careful calculation of the decoding failure rate for different classifications of error vectors. For most decoding procedures, a decoding failure error occurs when it takes more than a certain number of rounds. Because of this, a similar attack can be performed if an adversary is able to know the number of rounds it took for the decoding algorithm to succeed. This means that the GJS attack can be reformulated as a very strong side-channel attack when the decoding algorithm is not constant time.

While this attack is not at all relevant when the KEM is used ephemerally, it is still certainly best cryptographic practice to make sure that the decoder is constant time. This also ensures that if the generator matrix and parity check matrix are accidentally reused, a security vulnerability in the scheme only occurs if a decoding failure occurs.

4.4 Design Rationale

Design requirements: To establish an IND-CPA secure code-based ephemeral key encapsulation mechanism that has reasonable speeds and efficiency, while using compact public keys.

Design rational and important decisions: The QC-MDPC KEM is based on the design from Table 4 of [9]. The design allows for the masking of the plaintext x (which is normally revealed), and limits an attacker's control over the choice of error vector e. However, the QC-MDPC McEliece encryption scheme is not perfectly compatible with the framework from [9] due to both the non-deterministic nature of the scheme and the looming threat of the GJS attack. As such, we designed the scheme to be a secure and efficient ephemeral KEM that is resistant to the GJS attack under "accidental key reuse".

5. Security of the QC-MDPC McEliece KEM

5.1 Theoretical Security Analysis

5.1.1 Game Definitions

Game 1 (IND-CPA)**.**

1. *$\mathcal{B}$ generates a public/private key pair $(G, H) \leftarrow \mathsf{QCMDPC.KeyGen}(n, w, r, t)$ and sends the public key G to $\mathcal{A}$.*
2. *$\mathcal{B}$ Generates a challenge by running $(K_0, C) \leftarrow \mathsf{KEM.Encap}(G, s)$ for a uniformly random seed s. They then generate a uniformly random K_1 and a uniformly random bit $b \xleftarrow{\$} \{0, 1\}$. $\mathcal{B}$ sends (K_b, C) to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs a guess b' for b.*

$\mathcal{A}$ wins Game 1 if $b = b'$.
Game 2 (OW-CPA)**.**

1. *$\mathcal{C}$ generates a public/private key pair $(G, H) \leftarrow \mathsf{QCMDPC.KeyGen}(n, w, r, t)$ and sends G to $\mathcal{B}$.*
2. *$\mathcal{C}$ chooses a uniform $x \leftarrow \{0, 1\}^k$ and a random error vector e, and computes and sends $c = \mathsf{QCMDPC.Encrypt}(G, x, e)$ to $\mathcal{B}$.*
3. *$\mathcal{B}$ performs some computation, then submits a (x', e') as a guess for a decryption of c.*

$\mathcal{B}$ wins Game 2 if $\mathsf{QCMDPC.Encrypt}(x', e') = c$.

5.1.2 Reduction

We now define the reduction algorithm, $\mathcal{B}$. $\mathcal{B}$ plays Game 2, OW-CPA with a challenger $\mathcal{C}$. The reduction $\mathcal{B}$ uses an adversary $\mathcal{A}$ as a subroutine. They will simulate Game 1 with $\mathcal{A}$.

Game 3 (IND-CPA to OW-CPA reduction)**.**

1. *$\mathcal{B}$ receives a public key G from $\mathcal{C}$. $\mathcal{B}$ forwards this public key to $\mathcal{A}$ as the public key for the KEM.*
2. *$\mathcal{B}$ receives a target cipher text, c^* from $\mathcal{C}$. They generate a uniformly random $K \in \{0,1\}^n$, and a uniformly random $C_2 \in \{0,1\}^n$. $\mathcal{B}$ then sends $((c^*, C_2), K)$ to $\mathcal{A}$ as the challenge for Game 1.*
3. *$\mathcal{B}$ will monitor queries made by $\mathcal{A}$ to the random oracle KDF_1. Whenever a query is made with the input e', $\mathcal{B}$ checks to see if e is the error vector used to encrypt c. If so, $\mathcal{B}$ computes the corresponding plain text x' and outputs (x', e') to $\mathcal{C}$ as their guess for the decryption of $\mathcal{C}$.*
4. *If $\mathcal{A}$ submits a bit b' as a guess for the bit b, $\mathcal{B}$ simply aborts and outputs $\perp$.*

In what follows, q_{KDF_1} and q_ν are the number of queries $\mathcal{B}$ makes to KDF_1, and ν respectively.

Theorem 5.1.1. *If an adversary $\mathcal{A}$ can win Game 1 with probability $\frac{1}{2} + \epsilon$, then the reduction $\mathcal{B}$ wins Game 2 with probability $\epsilon - (q_{\mathsf{KDF}_1} + q_\nu)/2^k$.*

Proof. Note that for any C_1 there is at most one (x, e) such that

$$\mathsf{QCMDPC.Encrypt}(pk, x, e) = C_1.$$

This follows from the fact that the weight of the errors is chosen to make encryption injective. Then note that for any (x, e) there is at most one k-bit string s such that $(s \oplus \mathsf{KDF}_1(\nu(s)), \nu(s)) = (x, e)$. This is because if we have s, s' that both map to the same (x, e), then $\nu(s) = \nu(s')$, and so $\mathsf{KDF}_1(\nu(s)) = \mathsf{KDF}_1(\nu(s'))$, which means $s = x \oplus \mathsf{KDF}_1(\nu(s)) = x \oplus \mathsf{KDF}_1(\nu(s')) = s'$.

In Game 3, the challenge is prepared differently than how it is prepared in Game 1, and so we must establish that the adversary $\mathcal{A}$ cannot tell that the challenge they are sent is not correctly formatted.

The distribution of the challenge is entirely characterized by the joint distribution of four variables — (x, e, C_2, K_b). First note that the marginal distribution of each of these variables (i.e. in isolation) is uniform in both Game 1 and 3. In Game 3 these variables are entirely independent - each is generated uniformly and independent of each other. In Game 1 however,

these variables do have a dependence determined by $\mathsf{KDF}_1, \mathsf{KDF}_2$, and ν. Specifically, for any (x, e, C_2) corresponding to a challenge, it is the case that:

- $\nu(\mathsf{KDF}_1(e) \oplus x) = e$
- $\mathsf{KDF}_2(\mathsf{KDF}_1(e) \oplus x) = C_2 || K$.

Notice that these dependencies can only be checked by querying $s = \mathsf{KDF}_1(e) \oplus x$ to ν or KDF_2. Certainly these dependencies might be checked by finding $(x.e)$ and calculating $s = \mathsf{KDF}_1(e) \oplus x$. However in this case, the reduction will be completed, as the adversary has queried the corresponding error e to KDF_1, allowing $\mathcal{B}$ to win Game 2. However we also need to consider the possibility of $\mathcal{A}$ obtaining a copy of $s = \mathsf{KDF}_1(e) \oplus x$ without actually querying KDF_1 with e. To be specific, we need to upper bound the adversaries ability to query KDF_2 or ν with $s = \mathsf{KDF}_1(e) \oplus x$ without first querying $\mathsf{KDF}_1(e)$. Without having queried $\mathsf{KDF}_1(e)$, the adversary has no information about s, and so the only thing they can do is guess it by querying random strings. Therefore the probability that they query s to either oracle is bounded by $\delta = (q_{\mathsf{KDF}_1} + q_\nu)/2^k$.

The last thing we need to establish to finish the proof is that the adversary's advantage in winning Game 1 exactly corresponds to the ability of $\mathcal{B}$ to decrypt c^*. In other words, it corresponds to $\mathcal{A}$ querying the e associated with c^* to KDF_1. As established before, there is a unique s associated with c^* and KDF_1. The only way for an adversary to determine if the K given to them with the challenge cipher text is the correct decapsulation or not is to query that exact s to KDF_2. Without doing this, the adversary has no information, and so their probability of winning is at most $1/2$. So, the quantity ϵ corresponds precisely to the event that they are able to recover the unique s associated with c^*, given by the equation $s \leftarrow x \oplus \mathsf{KDF}_1(e)$.

Since we have already considered the probability that the adversary can query s to KDF_2 or ν without having first queried $\mathsf{KDF}_1(e)$, this s can only be generated by calculating $x \oplus \mathsf{KDF}_1(e)$ for the (x, e) corresponding to c^*. So the advantage is equal to the probability that the adversary queries KDF_1 with e. As this is how $\mathcal{B}$ will decrypt c^*, we have that

$$\Pr_{\text{Game 3}}[\mathcal{B} \text{ wins}] \geq \epsilon - \frac{q_{\mathsf{KDF}_1} + q_\nu}{2^k} \tag{5.1}$$

□

5.2 Analysis Against Known Attacks

There are no additional attacks to those described in Section 3.2 which this document need consider.

5.3 Parameter Selection and Expected Security

The suggested parameter sets are exactly those proposed in Section 3.3. In fact, assuming secure primitives, one can show that an adversary who can break the QC-MDPC KEM can break the QC-MDPC McEliece encryption scheme. Hence, the expected security level(s) of the KEM are no less than those proposed in Section 3.3.

6. Performance of the QC-MDPC McEliece KEM

6.1 Performance Analysis

The following are for the $r = 32771$ parameter set (see Section 3.3).

6.1.1 Platform

Benchmarks were run using "supercop-20170904" with a "Dual core Intel Core i7-7500U" CPU. Turbo Boost was disabled.

6.1.2 Time

Data obtained from SUPERCOP:

```
keypair_cycles - 131540379 129644053 131038872 131051885
enc_cycles - 20180017 20263392 19861833 20045657
dec_cycles - 229002269 227912081 230032997 230389473
```

Approximately 131 million cycles for key generation, 20 million cycles for encapsulation, and 230 million cycles for decapsulation.

6.1.3 Space

The public key is 4097 bytes this can be calculated straightforwardly by rounding the r parameter (which is in bits) up to the nearest byte. The encapsulated message is 8226 bytes which is twice the size of the public key plus the size of the confirmation hash. The sparse private is 548 bytes since it uses two bytes to store the location of each set bit and has weight w.

6.2 Advantages and Limitations

The QC-MDPC McEliece KEM has the advantages of having relatively compact keys and being built on a strong, well studied foundation. However, there are also disadvantages to this KEM. For example, the protocol may not be fast enough for certain applications, and given the current level of analysis of the GJS attack, the KEM does not seem to be suitable for static key establishment in a general setting.

Bibliography

[1] Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, December 2016. URL http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf.

[2] Carlisle M. Adams and Henk Meijer. *Security-Related Comments Regarding McEliece's Public-Key Cryptosystem*, pages 224–228. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988. ISBN 978-3-540-48184-3. doi: 10.1007/3-540-48184-2_20. URL http://dx.doi.org/10.1007/3-540-48184-2_20.

[3] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. *arXiv preprint arXiv:1603.09383*, 2016.

[4] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 520–536. Springer, 2012.

[5] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[6] Daniel J. Bernstein. *Grover vs. McEliece*, pages 73–80. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-12929-2. doi: 10.1007/978-3-642-12929-2_6. URL http://dx.doi.org/10.1007/978-3-642-12929-2_6.

[7] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight Bounds on Quantum Searching. *Fortschritte Der Physik*, 46:493–505, 1998.

[8] A. Carleial and M. Hellman. A note on wyner's wiretap channel (corresp.). *IEEE Trans. Inf. Theor.*, 23(3):387–390, September 2006. ISSN 0018-9448. doi: 10.1109/TIT.1977.1055721. URL `http://dx.doi.org/10.1109/TIT.1977.1055721`.

[9] Alexander W Dent. A designer's guide to KEMs. *Lecture notes in computer science*, pages 133–151, 2003.

[10] Ilya Dumer. On minimum distance decoding of linear codes. In *Workshop Info. Theory*, volume 1, pages 50–52, 1991.

[11] Richard Durstenfeld. Algorithm 235: random permutation. *Communications of the ACM*, 7(7):420, 1964.

[12] Cédric Faure and Lorenz Minder. Cryptanalysis of the mceliece cryptosystem over hyperelliptic codes. In *Proceedings of the 11th international workshop on Algebraic and Combinatorial Coding Theory, ACCT*, volume 2008, pages 99–107, 2008.

[13] Ronald Aylmer Fisher, Frank Yates, et al. Statistical tables for biological, agricultural and medical research. pages 26–27, 1938.

[14] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[15] JK Gibson. Severely denting the gabidulin version of the mceliece public key cryptosystem. *Designs, Codes and Cryptography*, 6(1):37–45, 1995.

[16] Keith Gibson. The security of the gabidulin public key cryptosystem. In *Advances in CryptologyâĂŤEUROCRYPTâĂŹ96*, pages 212–223. Springer, 1996.

[17] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying Grover's algorithm to AES: quantum resource estimates. In *International Workshop on Post-Quantum Cryptography*, pages 29–43. Springer, 2016.

[18] Lov K. Grover. Quantum Mechanics Helps in Searching for a Needle in a Haystack. *Phys. Rev. Lett.*, 79(2):325–328, 1997.

[19] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors, 2016. Cryptology ePrint Archive, Report 2016/858.

[20] Ghazal Kachigar and Jean-Pierre Tillich. Quantum information set decoding algorithms. Cryptology ePrint Archive, Report 2017/213, 2017. `http://eprint.iacr.org/2017/213`.

[21] Phillip Kaye, Raymond Laflamme, and Michele Mosca. *An Introduction to Quantum Computing*. Oxford, 2007.

[22] Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' approach to authenticated diffie-hellman and its use in the IKE protocols. In *Annual International Cryptology Conference*, pages 400–425. Springer, 2003.

[23] C. Lavor, L. R. U. Manssur, and R. Portugal. Grover's Algorithm: Quantum Database Search, 2003. `arXiv:quant-ph/0301079v1`.

[24] P. J. Lee and E. F. Brickell. *An Observation on the Security of McEliece's Public-Key Cryptosystem*, pages 275–280. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[25] Bernhard MJ Leiner. LDPC codes–a brief tutorial. *Apr*, 8:1–9, 2005.

[26] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*. Elsevier, 1977.

[27] Tsutomu Matsumoto and Hideki Imai. Public quadratic polynominal-tuples for efficient signature-verification and message-encryption. In *Eurocrypt*, volume 88, pages 419–453. Springer, 1988.

[28] Ingo Von Maurich, Tobias Oder, and Tim Güneysu. Implementing QC-MDPC mceliece encryption. *ACM Transactions on Embedded Computing Systems (TECS)*, 14(3):44, 2015.

[29] Alexander May, Alexander Meurer, and Enrico Thomae. *Decoding Random Linear Codes in*, pages 107–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-25385-0. doi: 10.1007/978-3-642-25385-0_6. URL `http://dx.doi.org/10.1007/978-3-642-25385-0_6`.

[30] R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory, 1978.

[31] Robert McEliece. *The theory of information and coding*, volume 3. Cambridge University Press, 2002.

[32] Lorenz Minder and Amin Shokrollahi. Cryptanalysis of the sidelnikov cryptosystem. *Advances in Cryptology-EUROCRYPT 2007*, pages 347–360, 2007.

[33] Rafael Misoczki. *Two Approaches for Achieving Efficient Code-Based Cryptosystems*. Theses, Université Pierre et Marie Curie - Paris VI, November 2013. URL `https://tel.archives-ouvertes.fr/tel-00931811`.

[34] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, and Paulo S. L. M. Barreto. MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes, 2012. Cryptology ePrint Archive, Report 2012/409.

[35] Raphael Overbeck and Nicolas Sendrier. Code-based cryptography. In *Post-quantum cryptography*, pages 95–145. Springer, 2009.

[36] Jacques Patarin. Cryptanalysis of the matsumoto and imai public key scheme of eurocrypt'88. In *Crypto*, volume 95, pages 248–261. Springer, 1995.

[37] Christiane Peters. *Curves, Codes, and Cryptography*. PhD thesis, Technische Universiteit Eindhoven, the Netherlands, 2011. `http://alexandria.tue.nl/extra2/711052.pdf`.

[38] Albrecht Petzoldt, Stanislav Bulygin, and Johannes Buchmann. Selecting parameters for the rainbow signature scheme. *Post-Quantum Cryptography*, pages 218–240, 2010.

[39] E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, September 1962. ISSN 0096-1000. doi: 10.1109/TIT.1962.1057777.

[40] T. R. N. Rao and Kil-Hyun Nam. *Private-Key Algebraic-Coded Cryptosystems*, pages 35–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987. ISBN 978-3-540-47721-1. doi: 10.1007/3-540-47721-7_3. URL `http://dx.doi.org/10.1007/3-540-47721-7_3`.

[41] Eleanor G. Rieffel and Wolfgang Polak. An Introduction to Quantum Computing for Non-Physicists. *ACM Comput. Surv.*, 32(3):300–335, 2000.

[42] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[43] Miklós Sántha. Quantum walk based search algorithms. In *Proceedings of the 5th international conference on Theory and applications of models of computation*, (TAMC'08), pages 31–46, 2008.

[44] Nicolas Sendrier. Decoding One Out of Many. Cryptology ePrint Archive, Report 2011/367, 2011. URL `http://eprint.iacr.org/2011/367`.

[45] Vladimir M Sidelnikov and Sergey O Shestakov. On insecurity of cryptosystems based on generalized reed-solomon codes. *Discrete Mathematics and Applications*, 2(4):439–444, 1992.

[46] Jacques Stern. A method for finding codewords of small weight. In *Proceedings of the 3rd International Colloquium on Coding Theory and Applications*, pages 106–113, London, UK, UK, 1989. Springer-Verlag. ISBN 3-540-51643-3. URL `http://dl.acm.org/citation.cfm?id=646721.702702`.

[47] Robert M. Tanner. A recursive approach to low complexity codes. *IEEE Transactions on information theory*, 27(5):533–547, 1981.

[48] Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43(6):1757–1766, 1997.

A. KEM Options

This appendix describes some optional features of the QC-MDPC KEM which, if used, allow for additional flexibility. These options are fairly simple and can improve the KEM performance, improve security properties, or make it more suitable for use in certain contexts.

A.1 Key Confirmation Value C_2

The value C_2 is used to ensure the authenticity of the ciphertext. It should be noted that the error vector also performs this function — as part of decapsulation, $\nu(s)$ is checked against e. It is not clear if this check with C_2 is needed in all contexts for full security. It is possible that this check can be omitted to save on communication (and simplify the scheme). The KEM's proof of IND-CPA security still works if C_2 is removed. However, the proof is in the random oracle model and *not* the quantum random oracle model. In the quantum random oracle model, proofs of security in strong models such as IND-CPA or IND-CCA2 security have thus far typically required an 'additional hash', which C_2 could possibly serve as. The most conservative approach for post-quantum security is likely to leave it, but as research into conditions of IND security in the quantum random oracle model develops, it may be seen as redundant.

A.2 Appending Additonal Hash Information

The QC-MDPC KEM makes use of two external functions: the error generator function ν and the key derivation function KDF. Additionally, the KDF is used in two 'modes' depending on context and desired output length, KDF_1 and KDF_2. This can be achieved in several ways, the most natural of which is to simply append some information (for example an encoding of 1 or 2) in order to have two distinct KDF functions. This section describes how else

to modify the usage of the KEM by adding additional information to these functions; thereby impacting security and performance.

One technique is to append the public key to the input of these functions; when querying ν, KDF_1, or KDF_2, one can also concatenate the receiver's public key pk to the end of the input. This provides some generic advantages, as any attacks on the scheme related to these functions must now 'restart' in some sense for each user.

For example, consider the GJS attack as launched against the KEM. The attack is somewhat mitigated by the fact that the error vector must be generated by running a seed through the pseudo-random function ν. While a malicious adversary cannot choose an error to send as part of the encapsulation, they may be able to use post-selection in order to generate an error that is far from uniformly random. By requiring the additional concatenation of the public key in ν, an adversary attempting to attack users in this way must 'relaunch' the attack in some sense for each user.

By concatenating pk for the ν and KDF_1 functions, but *not* KDF_2, one could use this KEM for broadcast encryption. This would allow a user to encapsulate a single key, but send it to multiple people with independent encapsulation ciphertexts.

Concatenating information can also be used to eliminate the GJS attack and allow for IND-CCA2 secure static key use. We call this variant of the KEM ParQ. By repeatedly encapsulating a key through multiple encryptions, one can reduce the decoding failure rate to a level negligible in the security parameter. This removes the possibility of performing the GJS attack, or any possible improvement that takes advantage of decoding failures. Furthermore, this does not alter the parameters of the underlying QC-MDPC encryption scheme.

The key generation algorithm is the same as just $\mathsf{QCMDPC.KeyGen}$.

Algorithm 9 ParQ Encapsulation

Input: Public key pk, a seed $s \in \{0,1\}^r$.
Output: Session key K, key encapsulation $C = (C_1, \ldots, C_P)$.

1: **for** $i = 1$ to P **do**
2: Let $e_i = \nu(s||i)$.
3: Compute $x_i = s \oplus \mathsf{KDF}_1(e_i||i)$.
4: Compute $C_i = \mathsf{QCMDPC.Encrypt}(pk, x_i, e_i)$.
5: **end for**
6: Compute $K = \mathsf{KDF}_2(s)$.
7: Return session key K, key encapsulation $C = (C_1, \ldots, C_P)$.

Algorithm 10 ParQ Decapsulation

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: Set $i = 1$.
2: Run $(x_i, e_i) \leftarrow \mathsf{QCMDPC.Decrypt}(sk, C_i)$.
3: **if** QCMDPC.Decrypt resulted in a decoding failure **then**
4: Increment i and return to step 2. If $i = P$, return $\perp$.
5: **end if**
6: Compute $s = x_i \oplus \mathsf{KDF}_1(e_i||i)$.
7: Compute $K, C' = (C'_1, C'_2, \ldots, C'_P)$ from ParQ Encapsulation with seed s and public key pk.
8: **if** $C_i = C'_i$ for all $i \in \{1, \ldots, P\}$ **then**
9: Return K.
10: **else**
11: Return decapsulation failure $\perp$.
12: **end if**

The impact of decoding failures can be eliminated with this KEM. A decapsulation failure only happens if a decoding failure occurs for each separate ciphertext. Given that the base error rate is on the order of 10^{-7}, a decapsulation failure only happens with probability 10^{-7P}. For values of P as low as 12, this reduces the error rate to a level negligible in the security parameter.

This KEM achieves full IND-CCA2 security. The proof of this is to be published soon. The proof is shown in a model that fully considers decoding failures, so that no attack on the KEM may take advantage of these decoding failures. It reduces the security of the ParQ KEM to the OW-CPA security of QC-MDPC McEliece.

This KEM does not change the key sizes at all, only encapsulation sizes. Encapsulation sizes do increase by a factor of P, but it allows for static-key usage without concern for the GJS attack.