Error Correcting Codes in Post-Quantum Cryptography

by

John-Marc Desmarais

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Science

Ottawa Carleton Institute of
Mathematics and Statistics

Department of Mathematics
Carleton University
Ottawa, Ontario, Canada
August 2020

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Error Correcting Codes in Post-Quantum Cryptography

Submitted by **John-Marc Desmarais**
in partial fulfilment of the requirements for the degree of
Master of Science

D. Panario, Supervisor

G. Walsh

S. Wang

P. Mezo, Department Chair

Carleton University
2020

Abstract

This thesis gives an overview of the currently most mature key encapsulation mechanisms (KEMs) based on the theory of error correcting codes. It includes an introduction to the theory of error correcting codes in so much as it applies to these systems and how it can be used to encapsulate keys through a public key (PK) cryptosystem.

In order to add context to the KEMs, first the required basics of coding theory and a selection of some of the most common error correcting codes are covered. Then, we revisit public key cryptosystems, key encapsulation, and the security threat models that are being used. This is followed by a thorough description of the current NIST candidates for KEM using post-quantum cryptography: Classic McEliece, BIKE, LEDAcrypt, and HQC. We do not include rank metric methods such as ROLLO and RQC, which were NIST candidates until the second round, since they involve different features than those studied in this thesis.

The thesis is intended as a survey of current methods being used in this field. We also establish some of the problems which may pose interesting for further research.

Acknowledgments

I would like to acknowledge my thesis supervisor Daniel Panario for his assistance and guidance through several drafts of this thesis and my wife Nicole Cuillerier for her patience and support throughout the writing process.

Thanks also to my thesis committee Gary Walsh and Steven Wang for their insight, questions and suggestions for edits and improvements and to Colin Ingalls for chairing the thesis defence.

Special thanks to Nicki Gaertner for her assistance in guiding me through tricky administrative details from registration in the Master's program to thesis upload.

Table of Contents

Chapter 1

Introduction

1.1 Historical Context

The basic idea of error correcting codes pioneered by Hamming in 1950 [36] is based on the idea of adding redundant bits to a message in order to detect and correct transmission errors. The subsequent decades saw improvements on this code system, Bose–Chaudhuri–Hocquenghem (BCH) codes [20,37] of 1960, and Goppa codes [13,32] of 1970 being particularly relevant.

In 1962, Gallager [30] in his PhD thesis came up with a new form of error correcting code that was based on Low-Density matrices using a Bit-Flipping algorithm for decoding. This was independently rediscovered in 1996 by MacKay and Neal [45], and used to develop low-density parity check (LDPC) and medium-density parity check (MDPC) codes.

In 1976, Diffie and Hellman published the idea of public key or asymmetric cryptography. Until this time cryptography was based on the idea of symmetric cryptography in which the sender and receiver had to have a shared secret key in order to communicate securely. Asymmetric cryptography allows a sender to encrypt a message with a published public key which could only be decrypted by someone who had a private associated key. The first schemes based on this message were RSA [59] and McEliece [47] both published in 1978. RSA based its security on the hardness of factoring large numbers and McEliece based his security on the problem of decoding a random Goppa code.

RSA was chosen over McEliece as one of the most popular asymmetric cryptography schemes of the past 40 years, mainly due to its small public key sizes by comparison.

In 1994, Shor [62] came up with a few algorithms for quantum computers that made the previously computationally infeasible problems of factoring large numbers and finding discrete logarithms of numbers in modular rings suddenly feasible. In the subsequent decades, this did not have much effect on the popularity of RSA as a standard for asymmetric encryption. But, now in the advent of quantum computers of increasing bit sizes, this problem has returned. At the same time, the large key sizes required by code-based systems are no longer as much of an issue due to increasing access to high speed networks.

In 2016, the National Institute of Standards and Technology (NIST) published a call for proposals for post-quantum cryptosystems that would not be based on the hardness of factoring large numbers or of finding discrete logarithms. This thesis provides a thorough description of the Round 2 Candidates of this competition which are based upon coding theory. These are Classic McEliece, BIKE, LEDAcrypt, and HQC.

1.2 Thesis Structure

Chapter 2 covers coding theory as is necessary to understand the current state of the art in code-based cryptography. It begins with a description of the theory of error correcting codes and then describes several of the most popular code systems currently in use including Hamming Codes, BCH codes, Low-Density Parity-Check (LDPC) codes, Medium-Density Parity-Check (MDPC) codes, and Quasi-cyclic (QC) codes.

Chapter 3 introduces public key cryptosystems and how coding theory fits into this paradigm. It then presents the leading contenders for code-based KEM in Round 3 of the NIST standardization competition, announced in July 2020, including Classic McEliece, BIKE, and HQC, as well as second round contender LEDAcrypt. For each of these proposals, we describe in detail the problem on which the security is based, the system parameters, key generations, encapsulation, decapsulation, attacks on the system, and a security analysis.

Chapter 4 concludes with a summary of the findings and includes some potential areas for future research.

Chapter 2

Error Correcting Codes

2.1 Coding Theory

Definition 2.1.1 ([41, p.473])**.** Let H be an $(n-k) \times n$ matrix of rank $n-k$ with entries in $\mathbb{F}_q$. The set $\mathcal{C}$ of all n-dimensional vectors $\mathbf{c} \in \mathbb{F}_q^n$ such that $H\mathbf{c}^T = 0$ is a *linear* (n, k) *code* over $\mathbb{F}_q$; n is the *length* and k the *dimension* of the code. The elements of $\mathcal{C}$ are *codewords* (or *code vectors*), the matrix H is a *parity-check matrix* of $\mathcal{C}$. If $q = 2$, $\mathcal{C}$ is a *binary code*. If H is of the form (A, I_{n-k}), then $\mathcal{C}$ is a *systematic code*.

In this thesis, k represents the number of bits of data to be transmitted, n represents the length of the transmission, and $n-k$ is therefore the number of redundant bits added to a message for error detection and correction.

Definition 2.1.2 ([41, p.474])**.** The $k \times n$ matrix $G = (I_k, -A^T)$ is the *canonical generator matrix* of a linear (n, k) code with parity-check matrix $H = (A, I_{n-k})$.

This allows us to define an encoder for $\mathcal{C}$

$$f_G : \mathbb{F}_q^k \to \mathcal{C} \subseteq \mathbb{F}_q^n$$
$$x \mapsto xG.$$

Using the generator matrix, we convert a message $m \in \mathbb{F}_q^n$ into a codeword $c \in \mathbb{F}_q^n$. When the generator matrix is in canonical form, for a binary code, this corresponds to adding redundancy of $n-k$ bits to the end of the message.

Definition 2.1.3 ([41, p.474])**.** Let $\mathbf{x}, \mathbf{y}$ be two vectors in $\mathbb{F}_q^n$.

(i) The *Hamming distance* $d(\mathbf{x}, \mathbf{y})$ between $\mathbf{x}$ and $\mathbf{y}$ is the number of coordinates in which $\mathbf{x}$ and $\mathbf{y}$ differ.

(ii) The *Hamming weight* $w(\mathbf{x})$ of $\mathbf{x}$ is the number of nonzero coordinates of $\mathbf{x}$.

Lemma 2.1.4 ([41, p.474])**.** The Hamming distance is a metric on $\mathbb{F}_q^n$, that is for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$, we have

(i) $d(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$;

(ii) $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$;

(iii) $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$.

Definition 2.1.5 ([41, p.475])**.** The number

$$d_c = \min_{\substack{\mathbf{u},\mathbf{v}\in\mathcal{C} \\ \mathbf{u}\neq\mathbf{v}}} d(\mathbf{u}, \mathbf{v}) = \min_{0\neq\mathbf{c}\in\mathcal{C}} w(\mathbf{c})$$

is the *minimum distance* of the linear code $\mathcal{C}$.

The minimum distance between any two codewords is the same as the minimum of the weights of all the codewords.

Using the norm of a vector space, these concepts of distance and weight can be extended to systems that use other metrics.

Definition 2.1.6 (Norm on a Vector Space [58, p.28])**.** A *norm* on a vector space V is any function $\| \ \|: V \to \mathbb{R}$ with the three properties of distance, namely if $v, w \in V$ and $\lambda \in \mathbb{R}$, then

$$\begin{aligned} &\| \ v \ \| \geq 0 \text{ and } \| \ v \ \| = 0 \text{ if and only if } v = 0, \\ &\| \ \lambda v \ \| = |\lambda| \ \| \ v \ \|, \\ &\| \ v + w \ \| \leq \| \ v \ \| + \| \ w \ \| \ . \end{aligned}$$

The unqualified term "norm" typically refers to the L^2-norm, that is,

$$\| \ \mathbf{x} \ \| = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

Other norms, for example those based on the Hamming distance, meet the three distance properties above and are used in coding theory.

Definition 2.1.7 ([48, p.8]). Let $\mathcal{R}$ be the ring defined by $\mathcal{R} = \mathbb{F}_2[x]/(x^n - 1)$, let $\mathcal{C}$ be an $[n, k]$ linear code over $\mathcal{R}$ and let ω be a norm on $\mathcal{R}$. The *minimum distance* of $\mathcal{C}$ is

$$d_c = \min_{\substack{\mathbf{u},\mathbf{v}\in\mathcal{C} \\ \mathbf{u}\neq\mathbf{v}}} \omega(\mathbf{u} - \mathbf{v}).$$

Definition 2.1.8 ([41, p.476]). Let H be the parity-check matrix of a linear (n, k) code $\mathcal{C}$. Then the vector $S(\mathbf{y}) = H\mathbf{y}^T$ of length $n - k$ is the *syndrome* of $\mathbf{y}$.

Theorem 2.1.9 (Syndrome Properties [41, p.477]). For $\mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$ we have

(i) $S(\mathbf{y}) = 0$ if and only if $\mathbf{y} \in \mathcal{C}$;

(ii) $S(\mathbf{y}) = S(\mathbf{z})$ if and only if $\mathbf{y} + \mathcal{C} = \mathbf{z} + \mathcal{C}$.

Proof.

(i) Follows immediately from the definition of $\mathcal{C}$ in terms of H.

(ii) Follows from $S(\mathbf{y}) = S(\mathbf{z})$ if and only if $H\mathbf{y}^T = H\mathbf{z}^T$ if and only if $H(\mathbf{y} - \mathbf{z}) = 0$ if and only if $\mathbf{y} - \mathbf{z} \in \mathcal{C}$ if and only if $\mathbf{y} + \mathcal{C} = \mathbf{z} + \mathcal{C}$. □

A vector $\mathbf{y} \in \mathbb{F}_q^n$ received by the system can be checked against the parity-check matrix by calculating $S(\mathbf{y}) = H\mathbf{y}^T$ and if $S(\mathbf{y}) = 0$ then either there are no errors in the transmission, or there is a greater number of errors in the system than can be determined using H. In either case, $S(\mathbf{y}) = 0$ means to us that y is a codeword.

On the other hand, if an error $\mathbf{e}$ has been added to the transmitted code then, for $\mathbf{c} \in \mathcal{C}$, we have the received message $\mathbf{y} = \mathbf{c} + \mathbf{e}$. Then,

$$\begin{aligned} S(\mathbf{y}) &= S(\mathbf{c} + \mathbf{e}) = S(\mathbf{c}) + S(\mathbf{e}) \\ &= 0 + S(\mathbf{e}) = S(\mathbf{e}). \end{aligned}$$

This leads naturally to the idea of error vectors.

Definition 2.1.10 (Error vector [41, p.474]). If $\mathbf{c}$ is a codeword and $\mathbf{y}$ is the received word, then $\mathbf{e} = \mathbf{y} - \mathbf{c}$ is the *error vector*.

The driving principle of coding theory is to be able to remove a nonzero error vector from the received word in order to rebuild the original codeword that was transmitted.

Definition 2.1.11 (Nearest Neighbour Decoding Principle [41, p.475])**.** The *nearest neighbour decoding principle* says that the codeword $\mathbf{c}$ with the smallest Hamming distance $d(\mathbf{c}, \mathbf{y})$ is most likely to be the transmitted word.

Theorem 2.1.12 ([41, p.475])**.** A code $\mathcal{C}$ with minimum distance $d_{\mathcal{C}}$ can correct up to t errors if $d_{\mathcal{C}} \geq 2t + 1$.

Proof. A ball $B_t(\mathbf{x})$ of radius t and center $\mathbf{x} \in \mathbb{F}_q^n$ consists of all the vectors $\mathbf{y} \in \mathbb{F}_q^n$ such that the distance $d(\mathbf{x}, \mathbf{y}) \leq t$. The nearest neighbour decoding rule ensures that each received word with t or fewer errors must be in a ball of radius t and centered at the transmitted codeword. To correct t errors, the balls with codewords as centers must not overlap. If $\mathbf{u} \in B_t(\mathbf{x})$ and $\mathbf{u} \in B_t(\mathbf{y})$ with $\mathbf{x},\ \mathbf{y} \in \mathcal{C}$ and $\mathbf{x} \neq \mathbf{y}$ then $d(\mathbf{x},\ \mathbf{y}) \leq d(\mathbf{x}, \mathbf{u}) + d(\mathbf{u}, \mathbf{y}) \leq 2t$ which is a contradiction to the minimum distance $d_{\mathcal{C}} \geq 2t + 1$. □

Lemma 2.1.13 ([41, p.475])**.** A linear code $\mathcal{C}$ with parity-check matrix H has minimum distance $d_{\mathcal{C}} \geq s + 1$ if and only if any s columns of H are linearly independent.

Proof. Assume there are s linearly dependent columns of H. Then if $H\mathbf{c}^T = 0$ and $w(\mathbf{c}) \leq s$ for suitable $\mathbf{c} \in \mathcal{C},\ \mathbf{c} \neq \mathbf{0}$, we have $d_{\mathcal{C}} \leq s$. Similarly, if any s columns of H are linearly independent, then there is no $\mathbf{c} \in \mathcal{C},\ \mathbf{c} \neq \mathbf{0}$ of weight not bigger than s, and hence $d_{\mathcal{C}} \geq s + 1$. □

Combining those two results gives that a t-error correcting code must have a minimum distance $d_{\mathcal{C}} \geq 2t + 1$ and we know that the parity check matrix must have s (where $d_{\mathcal{C}} \geq s + 1$) linearly independent columns. The Gilbert-Varshamov bound is based on these linearly independent columns.

Theorem 2.1.14 (Gilbert-Varshamov bound [41, p.480])**.** There exists a linear (n, k) code over $\mathbb{F}_q$ with minimum distance at least d whenever

$$q^{n-k} > \sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i.$$

Proof (Sketch). Construct a parity-check matrix H for this code with the first column being any nonzero $(n-k)$-tuple over $\mathbb{F}_q$, and the second column being any nonzero $(n-k)$-tuple over $\mathbb{F}_q$ that is linearly independent to the first column, that is, it is not

a scalar multiple of the first column. The columns continue to be picked so that the columns remain linearly independent. Then we count them to get

$$\sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i.$$

The resulting code has minimum distance at least d by our setup and Lemma 2.1.13. □

Definition 2.1.15 (Minimal Polynomial [50, p.18])**.** Let $K \subseteq F$ be fields, $\alpha \in F$ and $f(\alpha) = 0$ where f is a monic polynomial in $K[x]$. Then f is the *minimal polynomial* of α if α is not a root of any nonzero polynomial in $K[x]$ of lower degree.

For $K = \mathbb{F}_p$ and $F = \mathbb{F}_q$ where $q = p^m$, let $\alpha \in F$. Since $K \subseteq F$, the minimal polynomial of α over $\mathbb{F}_p$ is the lowest degree monic polynomial $M \in \mathbb{F}_p[x]$ such that $M(\alpha) = 0$.

Definition 2.1.16 (Cyclotomic Cosets [50, p.25])**.** The operation of multiplying by p divides the integers modulo $p^m - 1$ into *cyclotomic cosets* mod $(p^m - 1)$. The cyclotomic coset of s consists of $\{s,\ ps,\ p^2s, \ldots, p^{m_s-1}s\}$, where m_s is the smallest positive integer such that $p^{m_s}s \equiv s \pmod{p^m - 1}$.

It can be shown that elements from the same cyclotomic coset C have the same minimal polynomial and this polynomial is

$$M^{(i)}(x) = \prod_{j \in \mathcal{C}} (x - \alpha^j).$$

Example 2.1.17. As an example, for $\mathbb{F}_{2^4}$, $p = 2$, $m = 4$, and $p^m - 1 = 15$. The cyclotomic cosets are $C_0 = \{0\}$, $C_1 = \{1,\ 2,\ 4,\ 8\}$, $C_3 = \{3,\ 6,\ 12,\ 9\}$, $C_5 = \{5,\ 10\}$, and $C_7 = \{7,\ 14,\ 13,\ 11\}$. The associated minimum polynomials are

$$\begin{aligned}
M^{(0)}(x) &= x, \\
M^{(1)}(x) &= (x - \alpha^1)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8), \\
M^{(3)}(x) &= (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^9), \\
M^{(5)}(x) &= (x - \alpha^5)(x - \alpha^{10}), \\
M^{(7)}(x) &= (x - \alpha^7)(x - \alpha^{14})(x - \alpha^{13})(x - \alpha^{11}).
\end{aligned}$$

The precise algebraic reduction of these depends on the choice of irreducible used to define the polynomial field $\mathbb{F}_{2^4}$.

Definition 2.1.18 (Cyclic Code [40, p.318])**.** A linear (n, k) code $\mathcal{C}$ over $\mathbb{F}_2^q$ is *cyclic* if $(a_0, a_1, \ldots, a_{n-1}) \in \mathcal{C}$ implies $(a_{n-1}, a_0, \ldots, a_{n-1})$.

2.2 Selected Codes

Definition 2.2.1 (Hamming Code [41, p.478])**.** A binary code $\mathcal{C}_m$ of length $n = 2^m - 1, m \geq 2$, with an $m \times n$ parity check matrix H is a *binary Hamming code* if the columns of H are the binary representations of the integers $1, 2, \ldots, 2^m - 1$.

Example 2.2.2 ([41, p.478])**.** Consider $\mathcal{C}_3$. This gives length $n = 2^m - 1 = 7$ and produces the parity check matrix

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Upon receiving a word

$$\mathbf{y} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

we calculate the syndrome of $\mathbf{y}$, $S(\mathbf{y}) = H\mathbf{y}^T$. This gives us

$$S(\mathbf{y}) = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

which indicates that we have an error in the 5^{th} position. The received word can then be corrected by flipping the bit in position 5 to give the original message $\mathbf{y} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$. which, as expected has a syndrome of 0 and is, therefore, a codeword.

Bose–Chaudhuri–Hocquenghem (BCH) codes can be designed using minimal polynomials of Definition 2.1.15.

Definition 2.2.3 (BCH Codes [41, p.489])**.** Let b be a non-negative integer and let $\alpha \in \mathbb{F}_{q^m}$ be a primitive nth root of unity, where m is the multiplicative order of q modulo n. A *BCH code over* $\mathbb{F}_q$ *of length* n *and designed distance* $d, 2 \leq d \leq n$, is a cyclic code defined by the roots

$$\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+d-1}$$

of the generator polynomial.

If $M^{(i)}$ denotes the minimal polynomial of α^i over $\mathbb{F}_q$, then the generator polynomial g of a BCH code is of the form

$$g(x) = \text{lcm}(M^{(b)}(x), M^{(b+1)}(x), \ldots, M^{(b+d-2)}(x)).$$

If $b = 1$, the corresponding BCH codes are *narrow-sense* BCH codes. If $n = q^m - 1$, the BCH codes are *primitive*. If $n = q - 1$, a BCH code of length n over $\mathbb{F}_q$ is a *Reed-Solomon* code.

Theorem 2.2.4 (The BCH Bound [46, p.201])**.** Let $\mathcal{C}$ be a cyclic code with generator polynomial g such that for some integers $b \geq 0, \delta \geq 1$,

$$g(\alpha^b) = g(\alpha^{b+1}) = \cdots = g(\alpha^{b+\delta-2}) = 0,$$

that is, the code has a string of $\delta - 1$ consecutive powers of α as zeros. Then the minimum distance in the code is at least δ.

Proof. If $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ is in $\mathcal{C}$ then

$$c(\alpha^b) = c(\alpha^{b_1}) = \cdots = c(\alpha^{b+\delta-2}) = 0,$$

so that $H'c^T = 0$ where

$$H' = \begin{bmatrix} 1 & \alpha^b & \alpha^{2b} & \cdots & \alpha^{(n-1)b} \\ 1 & \alpha^{b+1} & \alpha^{2(b+1)} & \cdots & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{b+\delta-2} & \alpha^{2(b+\delta-2)} & \cdots & \alpha^{(n-1)(b+\delta-2)} \end{bmatrix}.$$

We observer that H' need not be the full parity check matrix for $\mathcal{C}$. Suppose c has weight $w \leq \delta - 1$. Then $H'c^T = 0$ implies

$$\begin{bmatrix} \alpha^{a_1 b} & \cdots & \alpha^{a_w b} \\ \alpha^{a_1(b+1)} & \cdots & \alpha^{a_w(b+1)} \\ \vdots & \ddots & \vdots \\ \alpha^{a_1(b+w-1)} & \cdots & \alpha^{a_w(b+w-1)} \end{bmatrix} \begin{bmatrix} c_{\alpha_1} \\ c_{\alpha_2} \\ \vdots \\ c_{\alpha_w} \end{bmatrix} = 0.$$

This implies that the determinant of the left matrix is zero, but its determinant is equal to

$$\alpha^{(a_1+\cdots+a_w)b} \cdot \det \begin{bmatrix} 1 & \cdots & 1 \\ \alpha^{a_1} & \cdots & \alpha^{a_w} \\ \vdots & \ddots & \vdots \\ \alpha^{a_1(w-1)} & \cdots & \alpha^{a_w(w-1)} \end{bmatrix},$$

which is the transpose of a Vandermonde matrix whose determinant is

$$\prod_{j=1}^{w-1} \prod_{i=j+1}^{w} (b_i - b_j) \neq 0$$

since all the $b_i = \alpha^{a_i}, 1 \leq i \leq w$ are distinct. The transpose must have a nonzero determinant and thus we have a contradiction. Therefore, $w(\mathbf{c}) \geq \delta$. Thus any $\delta - 1$ or fewer columns of H' are linearly independent and so, the minimum distance in the code is at least δ. □

In order to decode BCH codes, we can use the following algorithm.

Algorithm 2.2.5 (BCH Decoding [41, p.494])**.**

Input The received word $\mathbf{v}$ with at most t errors, a BCH code with designed distance $d \geq 2t + 1$.

Output The transmitted word $\mathbf{w}$ and the error $\mathbf{e}$.

1: Determine the syndrome of the received word $\mathbf{v}$,

$$S(\mathbf{v}) = (S_b, S_{b+1}, \ldots, S_{b+d-2})^T.$$

Let

$$S_j = \sum_{i=1}^{r} c_i \eta_i^j, \qquad b \le j \le b+d-2.$$

2: Determine the maximum number $r \le t$ such that the system of equations

$$S_{j+r} + S_{j+r-1}\tau_1 + \cdots + S_j \tau_r = 0, \qquad b \le j \le b+r-1,$$

in the τ_i has a non-singular coefficient matrix, thus obtaining the number r of errors that have occurred. Then set up the error-locator polynomial

$$s(x) = \prod_{i=1}^{r}(1-\eta_i x) = \sum_{i=0}^{r} \tau_i x^i.$$

Find the coefficients τ_i from the S_j.

3: Solve $s(x) = 0$ by substituting the powers of α (where α is a primitive nth root of unity) into $s(x)$. Thus find the error-location numbers η_i (this process is known as the Chien search).

4: Introduce the η_i in the first r equations of Step 1 to determine the error values c_i. Then find the transmitted word $\mathbf{w}$ from $w(x) = v(x) - e(x)$.

5: **return** $\mathbf{w}, \mathbf{e}$

There are many methods for performing Step 2. One suggested algorithm is Berlekamp-Massey [41, p.496]. Another method of decoding BCH codes is the Patterson's algorithm described later (Algorithm 2.3.8).

Definition 2.2.6 (Reed-Muller Codes [15, p.362])**.** The *shortened rth order generalized Reed-Muller code (GRM code)* of length $n = q^m - 1$ over $\mathbb{F}_q$ is the cyclic code whose generator polynomial g is defined by

$$g(x) = \prod_{\substack{j, \\ 0 \le w(j) < (q-1)m-r \\ 0 \le j < q^m-1}} (x - \alpha^j)$$

where $w(j)$ is the sum of the digits of the q-ary expansion of the integer j and α is a primitive element in $\mathbb{F}_{q^m}$.

The *rth-order GRM code* is obtained by annexing the all-one vector of length $N = q^m$ to the generator matrix of the shortened GRM code.

The *rth-order RM code* is the rth-order GRM code over $\mathbb{F}_2$.

Theorem 2.2.7 (RM Design Distance [15, p.362])**.** The rth order RM code is a subcode of the extended BCH code of designed distance $d = 2^{m-r}$.

Proof. For $j = 1, 2, 3, \ldots, 2^{m-r} - 2$, $w(j) < m - r$. □

Remark 2.2.8 ([48, p.32])**.** For any positive integers m and r with $0 \leq r \leq m$, there exists a binary rth- order Reed-Muller code denoted by $RM(r, m)$ with the following parameters

- code length $n = 2^m$;
- dimension $k = \sum_{i=0}^{r} \binom{m}{i}$;
- minimum distance $d_{min} = 2^{m-r}$.

Reed-Muller (Definition 2.2.6), Reed-Solomon Codes, BCH codes (Definition 2.2.3) and repetition codes are used in the Hamming Quasi-Cyclic cryptosystem, see Section 3.5.

Definition 2.2.9 (Alternant Codes [40, p.343])**.** Let $h_1, \ldots, h_n$ be arbitrary elements of $\mathbb{F}_q^*$ and let $\alpha_1, \ldots, \alpha_n$ be pairwise distinct elements of $\mathbb{F}_{q^m}$. Fix an integer t with $1 \leq t < n$. Then the *alternant code* $\mathcal{A}(\alpha, h)$ consists of all vectors in $\mathbb{F}_q^n$ that are in the null space of the $t \times n$ matrix.

$$H = \begin{bmatrix} h_1 & h_2 & \cdots & h_n \\ h_1\alpha_1 & h_2\alpha_2 & \cdots & h_n\alpha_n \\ h_1\alpha_1^2 & h_2\alpha_2^2 & \cdots & h_n\alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ h_1\alpha_1^{t-1} & h_2\alpha_2^{t-1} & \cdots & h_n\alpha_n^{t-1} \end{bmatrix}.$$

That is, if $H\mathbf{c}^T = 0$ then $\mathbf{c} \in \mathcal{A}(\alpha, h)$.

We observe that H can be written as the product of two matrices X and Y

$$H = XY = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \cdots & \alpha_n^{t-1} \end{bmatrix} \begin{bmatrix} h_1 & & & 0 \\ & h_2 & & \\ & & \ddots & \\ 0 & & & h_n \end{bmatrix}.$$

Theorem 2.2.10 (Maximum Distance of an Alternant Code [46, p.334])**.** An alternant code $\mathcal{A}(\alpha, h)$ has minimum distance $\delta \geq t + 1$.

Proof. Suppose $\mathbf{c}$ is a nonzero codeword of $\mathcal{A}(\alpha, h)$ with weight $w(\mathbf{c}) \leq t$. Then $H\mathbf{c}^T = XY\mathbf{c}^T = 0$. Set $\mathbf{b}^T = Y\mathbf{c}^T$, then $w(\mathbf{c}) = w(\mathbf{b})$ since Y is diagnonal and invertible. Thus $X\mathbf{b}^T = 0$, but X is a Vandermonde matrix whose determinant is nonzero since $\alpha_1, \ldots, \alpha_n$ are pairwise distinct, so $X\mathbf{b}^T \neq 0$. Hence we have a contradiction and $w(\mathbf{c}) \geq t + 1$. □

2.3 Goppa Codes

Consider the following alternative construction of a narrow sense BCH code.

Lemma 2.3.1 ([40, p.331])**.** Let $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$, then $\mathbf{c}$ is a codeword of the narrow sense BCH code over $\mathbb{F}_q$ defined by the roots $\alpha, \alpha^2, \ldots, \alpha^{d-1}$ of the generator polynomial if and only if

$$\sum_{i=0}^{n-1} c_i \alpha^{i(d-1)} \frac{x^{d-1} - \alpha^{-i(d-1)}}{x - \alpha^{-i}} = 0.$$

Proof. By definition, $(c_0, c_1, \ldots, c_{n-1})$ is a codeword of the given code if and only if

$$\sum_{i=0}^{n-1} c_i \alpha^{ij} = 0 \qquad 1 \leq j \leq d - 1. \tag{2.1}$$

And consider

$$\begin{aligned}\sum_{i=0}^{n-1} c_i \alpha^{i(d-1)} \frac{x^{d-1} - \alpha^{-i(d-1)}}{x - \alpha^{-i}} &= \sum_{i=0}^{n-1} c_i \alpha^{i(d-1)} \sum_{j=0}^{d-2} \alpha^{-i(d-2-j)} x^j \\ &= \sum_{j=1}^{d-1} \left(\sum_{i=0}^{n-1} c_i \alpha^{ij} \right) x^{j-1} \\ &= \sum_{j=1}^{d-1} (0)\, x^{j-1} = 0.\end{aligned}$$

The substitution in the last line comes from (2.1). The identity holds if and only if $(c_0, c_1, \ldots, c_{n-1})$ is a codeword. □

This lemma serves as a motivation for the definition of Goppa codes.

Definition 2.3.2 (Goppa Codes [28, p.157])**.** Let m and t be positive integers. Let

$$g(\mathbf{x}) = \sum_{i=0}^{t} g_i \mathbf{x}^i \in \mathbb{F}_{2^m}[\mathbf{x}]$$

be a monic *Goppa polynomial* of degree t, and

$$L = (\gamma_0, \ldots, \gamma_{n-1}) \in \mathbb{F}_{2^m}^n$$

the *code support*, a tuple of n pairwise distinct elements such that $g(\gamma_i) \neq 0$, for all $0 \leq i < n$.

For any vector $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in \mathbb{F}_2^n$, we define the *syndrome* of $\mathbf{c}$ by

$$S_c(\mathbf{x}) \equiv -\sum_{i=0}^{n-1} \frac{c_i}{g(\gamma_i)} \frac{g(\mathbf{x}) - g(\gamma_i)}{\mathbf{x} - \gamma_i} \pmod{g(\mathbf{x})}. \tag{2.2}$$

The *binary Goppa code* $G(L, g(\mathbf{x}))$ over $\mathbb{F}_2$ is the set of all $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in \mathbb{F}_2^n$ such that the identity

$$S_c(\mathbf{x}) = 0$$

holds in the polynomial ring $\mathbb{F}_{2^m}[\mathbf{x}]$, or equivalently if

$$S_c(\mathbf{x}) = \sum_{i=0}^{n-1} \frac{c_i}{\mathbf{x} - \gamma_i} \equiv 0 \pmod{g(\mathbf{x})}. \tag{2.3}$$

Thus, we have

$$\begin{aligned} G(L, g(\mathbf{x})) &= \{\mathbf{c} \in \mathbb{F}_2^n;\ S_c(\mathbf{x}) = 0\} \\ &= \{\mathbf{c} \in \mathbb{F}_2^n;\ S_c(\mathbf{x}) \equiv 0 \pmod{g(\mathbf{x})}\}. \end{aligned}$$

To produce the parity check matrix of a Goppa code $\Gamma(L, g)$, where $L = (\alpha_1, \ldots, \alpha_n)$ and g is a Goppa polynomial, we begin from Equation (2.3) of Definition 2.3.2. Slightly modified, this gives us

$$S_c(x) = \sum_{i=1}^{n} \frac{c_i}{x - \alpha_i} \equiv 0 \pmod{g(x)}. \tag{2.4}$$

Since $x - \alpha_i$ does not divide g in the ring of polynomials mod g, the inverse of $x - \alpha_i$ is

$$(x - \alpha_i)^{-1} = -\frac{g(x) - g(\alpha_i)}{x - \alpha_i} g(\alpha_i)^{-1}.$$

Indeed,

$$-(x - \alpha_i)\frac{(g(x) - g(\alpha_i)}{x - \alpha_i} g(\alpha_i)^{-1} \equiv 1 \pmod{g(x)}.$$

Therefore a is in $\Gamma(L, g)$ if and only if the polynomial.

$$\sum_{i=1}^{n} a_i \frac{g(x) - g(\alpha_i)}{x - \alpha_i} g(\alpha_i)^{-1} = 0.$$

If $g(x) = \sum_{i=0}^{t} g_i x^i$ for $g_i \in \mathbb{F}_{q^m}$ and $g_t \neq 0$ then,

$$\begin{aligned} \frac{g(x) - g(\alpha_i)}{x - \alpha_i} =& g_t(x^{t-1} + x^{t-2}\alpha_i + \cdots + \alpha_i^{t-1}) + \\ & g_{t-1}(x^{t-2} + \cdots + \alpha_i^{t-2}) + \cdots + g_2(x + \alpha_i) + g_1. \end{aligned}$$

Setting the coefficients $x^{t-1}, x^{t-2}, \ldots, 1$ to zero, we get that a is in $\Gamma(L, g)$ if and only

if $Ha^T = 0$, where

$$H = \begin{bmatrix} g_t g(\alpha_1)^{-1} & \cdots & g_t g(\alpha_n)^{-1} \\ (g_{t-1} + \alpha_1 g_t) g(\alpha_1)^{-1} & \cdots & (g_{t-1} + a_n g_t) g(\alpha_n)^{-1} \\ \vdots & \ddots & \vdots \\ (g_1 + \alpha_1 g_2 + \cdots \alpha_i^{t-1} g_t) g(\alpha_1)^{-1} & \cdots & (g_1 + \alpha_n g_2 + \cdots + \alpha_n^{t-1} g_t) g(\alpha_n)^{-1} \end{bmatrix}$$
$$= CYZ,$$

where

$$C = \begin{bmatrix} g_t & 0 & 0 & \cdots & 0 \\ g_{t-1} & g_t & 0 & \cdots & 0 \\ g_{t-2} & g_{t-1} & g_t & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \cdots & g_t \end{bmatrix},$$

$$Y = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{t-1} & \alpha_2^{t-1} & \cdots & \alpha_n^{t-1} \end{bmatrix},$$

$$Z = \begin{bmatrix} g(\alpha_1)^{-1} & & & 0 \\ & g(\alpha_2)^{-1} & & \\ & & \ddots & \\ 0 & & & g(\alpha_n)^{-1} \end{bmatrix}.$$

Since C above is invertible, another parity check matrix for the code is

$$H' = YZ = \begin{bmatrix} g(\alpha_1)^{-1} & \cdots & g(\alpha_n)^{-1} \\ \alpha_1 g(\alpha_1)^{-1} & \cdots & \alpha_n g(\alpha_n)^{-1} \\ \vdots & \ddots & \vdots \\ \alpha_1^{t-1} g(\alpha_1)^{-1} & \cdots & a_n^{t-1} g(\alpha_n)^{-1} \end{bmatrix}.$$

This H' formation is generally easier to work with.

A parity check matrix with elements from $\mathbb{F}_q$ is obtained by replacing each entry of H or H' by a length m column vector with elements from $\mathbb{F}_q$ [46, p.339].

Theorem 2.3.3. The dimension of a Goppa code $\Gamma(L, g)$ is at least $n - mt$ and its minimum distance is at least $t + 1$.

Proof. The parity check matrix for a Goppa code is

$$H' = XY = \begin{bmatrix} g(\alpha_1)^{-1} & \cdots & g(\alpha_n)^{-1} \\ \alpha_1 g(\alpha_1)^{-1} & \cdots & \alpha_n g(\alpha_n)^{-1} \\ \vdots & \ddots & \vdots \\ \alpha_1^{t-1} g(\alpha_1)^{-1} & \cdots & \alpha_n^{t-1} g(\alpha_n)^{-1} \end{bmatrix}.$$

Let $\alpha = \{\alpha_1, \ldots \alpha_n\}$ and $h = \{g(\alpha_1)^{-1}, \ldots, g(\alpha_n)\}$. Then we have that the Goppa code is an alternant code and hence by Theorem 2.2.10 the minimum distance is at least $t + 1$. □

Definition 2.3.4 (Binary Separable Goppa Code [46, p.342])**.** A binary Goppa code whose Goppa polynomial contains no roots in $\mathbb{F}_{2^m}$ and no repeated roots in any extension field, is a *binary separable Goppa code.*

Theorem 2.3.5 ([40, p.333])**.** For a binary separable Goppa code, the minimum distance is at least $2t + 1$.

Proof. If $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_2^n$ is a codeword of weight $w > 0$ in the binary Goppa code $\Gamma(L, g)$ then $c_{i_1} = c_{i_2} = \cdots = c_{i_w} = 1$ with $0 \leq i_1 < i_2 < \cdots < i_w \leq n-1$

with all other $c_i = 0$. If $L = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\} \subseteq \mathbb{F}_{2^m}$, define

$$f(x) = \prod_{j=1}^{w}(x - \gamma_{i_j}) \in \mathbb{F}_{2^m}[x].$$

From Definition 2.3.2 Equation (2.2) we obtain

$$\begin{aligned} 0 &= f(x)\sum_{i=0}^{n-1} c_i g(\gamma_i)^{-1}\frac{g(x) - g(\gamma_i)}{x - \gamma_i} = f(x)\sum_{j=1}^{w} g(\gamma_{i_j})^{-1}\frac{g(x) - g(\gamma_{i_j})}{x - \gamma_{i_j}} \\ &= \sum_{j=1}^{w} g(\gamma_{i_j})^{-1}(g(x) - g(\gamma_{i_j}))\prod_{\substack{h=1 \\ h\neq j}}^{w}(x - \gamma_{i_h}). \end{aligned}$$

Taking the last polynomial mod g we get

$$0 \equiv -\sum_{j=1}^{w}\prod_{\substack{h=1 \\ h\neq j}}^{w}(x - \gamma_{i_h}) \equiv -f'(x) \pmod{g(x)}.$$

Hence, g divides f'. Since we are in characteristic 2, f' contains even powers and is thus, the square of a polynomial in $\mathbb{F}_{2^m}[x]$. Since g is separable, it contains no multiple roots and so $g^2 \mid f'$. As a consequence,

$$w - 1 \geq \deg(f'(x)) \geq 2t.$$

That is, any nonzero codeword has weight of at least $2t + 1$. □

Example 2.3.6 ([40, p.334]). Let $g(x) = x^2 + x + 1$ and

$$L = \mathbb{F}_8 = \{0, 1, \alpha, \ldots, \alpha^2 + \alpha + 1\},$$

where α is a primitive element of $\mathbb{F}_8$ satisfying $\alpha^3 + \alpha + 1 = 0$.
We find the following parameters: $m = 3, t = 2, n = 8, k \geq n - mt = 2, d \geq 2t + 1$.

The intersection of $\mathbb{F}_2^n$ with the null space of the matrix $\Gamma(L, g)$ is

$$H = \begin{pmatrix} g(0)^{-1} & g(1)^{-1} & \cdots & g(\alpha^6)^{-1} \\ g(0)^{-1}\cdot 0 & g(1)^{-1}\cdot 1 & \cdots & g(\alpha^6)^{-1}\cdot\alpha^6 \end{pmatrix}.$$

Using the basis $\{1,\ \alpha,\ \alpha^2\}$ of $\mathbb{F}_8$ over $\mathbb{F}_2$, we get the binary matrix

$$H' = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

We obtain that H' is a parity check matrix for the linear $(8, 2)$-Goppa code $\Gamma(L, g)$ consisting of codewords $\{(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$, $(0\ 0\ 1\ 1\ 1\ 1\ 1\ 1)$, $(1\ 1\ 0\ 0\ 1\ 0\ 1\ 1)$, $(1\ 1\ 1\ 1\ 0\ 1\ 0\ 0)\}$

If the received message is not one of these codewords then there are errors and, in this case, we use Patterson's algorithm to find where the errors are located. Let $\mathbf{m} \in \mathbb{F}_2^n$ be a codeword, $\mathbf{e} \in \mathbb{F}_2^n$ with $w(\mathbf{e}) \leq t$, and $\mathbf{y} = \mathbf{m} \oplus \mathbf{e}$. We receive $\mathbf{y}$ and wish to compute $\mathbf{e}$ and $\mathbf{m}$. We know that Goppa codes are linear and so

$$S(\mathbf{y}) = S(\mathbf{m} + \mathbf{e}) = S(\mathbf{m}) + S(\mathbf{e}),$$

and we are given that $\mathbf{m}$ is a codeword so $S(\mathbf{m}) = \mathbf{0}$. This gives us

$$S(\mathbf{y}) \ = \ S(\mathbf{e}).$$

Using the syndrome of $\mathbf{e}$, we calculate the error-locator polynomial for the received message.

Definition 2.3.7. The *error-locator polynomial* $\sigma_e(X)$ is given by

$$\sigma_e(X) = \prod_{j \in T_e} (X - \gamma_j) \in \mathbb{F}_{2^m},$$

where $T_e = \{i : e_i = 1\}$.

Thus, for any bit j in the syndrome, we multiply together the factors $(X - \gamma_j)$.

Then, from Definition 2.3.2 Equation (2.3) it follows that

$$\sigma_e(X)S_e(X) \equiv \sigma'_e(X) \pmod{g(X)}. \tag{2.5}$$

Splitting $\sigma_e(X)$ into odd and even powers, we get

$$\sigma_e(X) = \alpha^2(X) + X\beta^2(X).$$

Since we are in $\mathbb{F}_2$, we have $\sigma'_e(X) = \beta^2(X)$, from (2.5) we get

$$\beta^2(X)(XS_e(X) + 1) \equiv \alpha^2(X)S_e(X) \pmod{g(X)}. \tag{2.6}$$

If $S_e(X) = 0$, there are no errors in the received message, so we can assume that that is not the case and there exists an inverse mod(g), call it $T(X) = S_e^{-1}$. Multiply this by (2.6), we get

$$\beta^2(X)(X + T(X)) \equiv \alpha^2(X) \pmod{g(x)}. \tag{2.7}$$

Since we have a square on the right side, the elements on the left must also be square thus $X + T(X)$ is a square. Let $v(X) = \sqrt{X + T(X)}$ then (2.7) can be rewritten as

$$\beta(X)v(X) \equiv \alpha(X) \pmod{g(x)}. \tag{2.8}$$

Given that the degree of $\sigma_e(X) \leq t$, apply the extended Euclidean algorithm to find $\alpha(X)$, $\beta(X)$ and $v(X)$ with the degree $\alpha(X) \leq \lfloor \frac{t}{2} \rfloor$ and degree $\beta(X) \leq \lfloor \frac{t-1}{2} \rfloor$. The fact that we can always find an $\alpha(X)$ and $\beta(X)$ with degree lower than these bounds is based on a result by Sugiyama et al. [67].

This computation of zeros for $\sigma_e(X) = \alpha^2(X) + X\beta^2(X)$ leads to the vectors $\mathbf{e}$ and hence, $\mathbf{m}$.

Algorithm 2.3.8 (Patterson's Algorithm [28, p.161])**.**

Input: $(\Gamma(L, g(x)))$: A binary irreducible Goppa code.

$(\mathbf{y} = \mathbf{m} + \mathbf{e})$: A received vector, where $\mathbf{m}$ is a codeword and $\mathbf{e}$ an error vector.

Output: $(\mathbf{m})$: The message.

$(\mathbf{e})$: The error vector.

Determine the syndrome of the received word $\mathbf{y}$,

$$S_y(x) \equiv \sum_{i=0}^{n-1} \frac{y_i}{x - \gamma_i} \pmod{g(x)}$$

Or, use the parity check matrix $S_y(x) = H\mathbf{y}^T$.

if $S_y(x) \equiv 0 \pmod{g(x)}$ **then**
    There is no error $\mathbf{m} = \mathbf{y}$
    return $(\mathbf{y}, 0)$
else
    There are errors, $\mathbf{c} = \mathbf{m} + \mathbf{e}$
    $T(x) = S_y^{-1}(x) \pmod{g(x)}$
    $\tau(x) = \sqrt{T(x) + x} \pmod{g(x)}$
    $i = 0$ ▷ /*Initialize variables to use the Extended Euclidean Algorithm. */
    $r_{-1}(x) = \alpha_{-1}(x) = g(x); \quad r_0(x) = \alpha_0(x) = \tau(x)$
    $\beta_{-1}(x) = 0; \quad \beta_0(x) = 1$
    while $\deg(r_i(x) \leq \lfloor (t+1)/2 \rfloor$ **do**
        $i = i + 1$
        Determine $q_i(x)$ and $r_i(x)$ such that $r_i(x) = r_{i-2}(x) - q_i(x)r_{i-1}(x)$ and $\deg(r_i(x)) < \deg(r_{i-1}(x))$
        $\beta_i(x) = \beta_{i-2}(x) + q_i(x)\beta_{i-1}(x)$
        $\alpha_i(x) = r_i(x)$
    end while
    $\sigma(x) = c^2((\alpha_i(x))^2 + x(\beta_i(x))^2)$ with $c \in \mathbb{F}_{2^m}$ such that $\sigma(x)$ is monic.
    for $i = 0$ **to** $n - 1$ **do** ▷ /* Determine the zeros of $\sigma_e(x)$. */
        if $\sigma(\gamma_i) = 0$ **then**
            $\mathbf{e}_i = 1$
        else
            $\mathbf{e}_i = 0$
        end if
    end for
     $\mathbf{m} = \mathbf{y} + \mathbf{e}$ **return** $(\mathbf{m}, \mathbf{e})$
end if

This algorithm was developed by Patterson [56]. This treatment is based on Englebert, Overbeck and Schmidt [28, p.161]

The extended Euclidean algorithm is used in Step 7 to compute $S_y^{-1}(x)$ (mod $g(x)$). Computing τ requires a linear mapping on $\mathbb{F}_{2^m}(x)/g$. Both operations take on the order of $\mathcal{O}(t^2m^2)$ operations [54, p.160].

Finding the roots of σ_e using the above algorithm (Steps 20-26) takes $n(tm^2+tm)$ binary operations [54, p.160] however, there are several alternative algorithms. A survey of factorization methods for polynomials over finite fields can be found in a paper of von zur Gathen and Panario [72]. These include the Berklekamp algorithm as well as other deterministic and probabilistic factorization algorithms.

2.4 Low-Density Parity Check and Medium-Density Parity Check (LDPC/MDPC) Codes

In his 1962 PhD work, Gallager [30] discovered LDPC codes which due to computational limits at the time were unfeasible for implementation and treated as a curiosity. In 1997, Neal and MacKay rediscovered LDPC codes [45] and showed many of the same results, this time the computational requirements were more feasible.

Definition 2.4.1 (Low Density Parity Check Codes [30, p.21])**.** A *Low-density parity-check (LDPC) code* $\mathcal{C}(n,k)$ is a code specified by a parity check matrix containing mostly 0's and only a small number of 1's. In particular, an (n,j,k) low-density code is a code of block length n with a matrix where each column contains a small fixed number j of 1's and each row contains a small fixed number k of 1's.

This parity check matrix can be used to describe a Tanner graph. A brief introduction to the representation of LDPC as graphs follows.

Definition 2.4.2 (Graphs [19, p.2])**.** A *graph* G is an ordered pair $(V(G), E(G))$ consisting of

- a set $V(G)$ of *vertices* and
- a set $E(G)$, disjoint from $V(G)$ of *edges* together with
- an *incidence function* ψ_G that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G.

If $e \in E(G)$ is an edge and $u, v \in V(G)$ are vertices such that $\psi_G(e) = u, v$ then e *joins* u and v, and the vertices u and v are the *ends* of e.

The number of vertices in $v(G) = |V(G)|$ is the *order* and the number of edges $e(G) = |E(G)|$ is the *size* of G.

Definition 2.4.3 (Vertex Degree [19, p.7])**.** The *degree* of a vertex $v \in V(G)$ denoted $d_G(v)$ is the number of edges of G that are incident with vertex v, i.e. the number of edges that are connected to vertex v.

Definition 2.4.4 (Bipartite Graph [19, p.4])**.** A graph is *bipartite* if its vertex set can be partitioned into two subsets X and Y so that every edge has one end in X and another end in Y. We denote a bipartite graph G with a bipartition (X, Y) by $G[X, Y]$.

Definition 2.4.5 (LDPC Code representation [6, p.9])**.** An LDPC code $\mathcal{C}(n, k)$ may be represented by a *Tanner graph*, a bipartite graph with the vertex set partitioned into n *variable* nodes and $r = n - k$ *parity check* nodes.

For example the matrix

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

consists of seven variable nodes $v_i,\ 0 \leq i < 7$ as columns and three parity-check nodes $c_j,\ 0 \leq j < 3$ as rows. An edge between v_i and c_j exists if the entry $h_i j = 1$.

Encoding using LDPC parity-check codes is identical to that of encoding in the codes we have looked at so far. Using Gaussian elimination, a generator matrix G is derived and using this, a codeword is generated as usual

$$\mathbf{c} = \mathbf{m}G,$$

where G is the generator matrix of the LDPC code, $\mathbf{m} \in \mathbb{F}_2^k$ is the message to be encoded and $\mathbf{c} \in \mathbb{F}_2^n$ is a codeword.

As is generally the case, the parity-check matrix H can be used to determine whether a received message $\mathbf{y}$ is a codeword by finding the syndrome, but the correction of errors uses the bit flipping algorithm.

Definition 2.4.6 (Bit Flipping Decoding Problem [45, p.1])**.** The decoding problem is to find the most probable vector $\mathbf{x}$ such that $H\mathbf{x} = 0$ in $\mathbb{F}_2$, with the likelihood of

x given by

$$\prod_n f_n^{(x_n)},$$

where

$$f_n^{(1)} = \frac{1}{(1+\exp(\frac{-2ay_n}{\sigma^2}))}$$

$$f_n^{(0)} = 1 - f_n^{(1)},$$

and y_n is the output of the channel at time n.

For the algorithm, $N(m) \equiv \{n : H_{mn} = 1\}$ represents the bits n participating in check m and $N(m)\backslash n$ is the set $N(m)$ excluding bit n. Similarly, $M(n) \equiv \{m : H_{mn} \equiv 1\}$ represents the set of checks that n participates in. The quantity $q_{mn}^{(x)}$ is the probability that bit n of **y** is x given information obtained by checks excluding m and $r_{mn}^{(x)}$ is the probability that check m is satisfied if bit n of **y** is fixed at x.

Algorithm 2.4.7 (Bit Flipping Algorithm [45, p.1])**.**

Input: (H): The parity check matrix, the rows of which represent the check equations.

($\mathbf{y} \in \mathbb{F}_2^n$): A received message consisting of n bits.

A maximum number of iterations for the main loop.

Output: ($\mathbf{m} \in \mathbb{F}_2^k$): The message

($\mathbf{e} \in \mathbb{F}_2^n$): The error vector .

1: Initialize $q_{mn}^{(0)} = f_n^{(0)}$ and $q_{mn}^{(1)} = f_n^{(1)}$. This represents the probability that bit n of vector **y** is 0 and 1, respectively.

2: **while** $H\mathbf{y}' \neq 0$ and we haven't reached max iterations **do**

3: Set $\delta q_{mn} = q_{mn}^{(0)} - q_{mn}^{(1)}$.

4: **for** each m, n **do** ▷ /* Horizontal step */

5: Compute

$$\delta r_{mn} = \prod_{n' \in N(m)\backslash n} \delta q_{mn'}.$$

6: Set

$$r_{mn}^{(0)} = \frac{1}{2}(1 + \delta r_{mn})$$

and

$$r_{mn}^{(1)} = \frac{1}{2}(1 - \delta r_{mn}).$$

7: **end for**

8: **for** each m, n and $x = 0, 1$ **do** ▷ /* Vertical Step */

9: Update

$$q_{mn}^{(x)} = \alpha_{mn} f_n^{(x)} \prod_{m' \in M(n) \setminus m} r_{m'n}^{(x)}$$

10: where α_{mn} is chosen such that $q_{mn}^{(0)} + q_{mn}^{(1)} = 1$.

11: Update psuedo-posterior probabilities $q_{mn}^{(0)}$ and $q_{mn}^{(1)}$ by

$$q_n^{(x)} = \alpha_n f_n^{(x)} \prod_{m \in M(n)} r_{mn}^{(x)}.$$

12: **end for**

13: Calculate $H\mathbf{y}'$ with updated values based on the new probabilities.

During step 1, the most probable set for any of the bits is $\mathbf{y}$. Thus, the variable nodes are set to the values from the received word. That is, for each n, $q_{mn}^{(1)}$ is set to y_n and $q_{mn}^{(0)}$ is set to the inverse $y_n \oplus 1$. During the horizontal step, the probability of each bit being set properly is calculated for n, usually by counting the parity of the checks that do not include n and if it is 1, this increases the probability that the desired bit is a 1. This is calculated for each bit y_n. Then during the horizontal step and based on some threshold α_{mn}, the array of guesses is updated by flipping some of the bits of $\mathbf{y}$. The syndrome is then checked. This continues until all the bits are set correctly (the syndrome is 0) or a maximum number of iterations is reached.

2.5 Quasi-Cyclic (QC) Codes

A quasi-cyclic code is a generalization of cyclic codes in Definition 2.1.18. While a cyclic shift by one position of any codeword in a cyclic code results in a codeword, in a quasi-cyclic code, any shift by n_0 positions of any codeword results in a codeword.

Definition 2.5.1 (Quasi-Cyclic Codes [3, p.2])**.** A *binary quasi-cyclic* (QC) code of index n_0 and order r is a linear code which admits as generator matrix a block-circulant matrix of order r and index n_0. A (n_0, k_0)-QC code is a quasi-cyclic code of index n_0, length $n_0 r$ and dimension $k_0 r$ [6, p.41].

Proposition 2.5.2 (Quasi-Cyclic Code Properties [6, p.23])**.** A quasi-cyclic code is defined as a linear block with dimension $k = pk_0$ and length $n = pn_0$ having the following properties.

(i) Each codeword is formed by a series of p blocks of n_0 symbols, each of which is formed by a series of p blocks of n_0 symbols followed by $n_0 - k_0$ redundancy symbols.

(ii) Each cyclic shift of a codeword by n_0 symbols yields another valid codeword.

Definition 2.5.3 (Binary Circulant Matrix [6, p.33])**.** A *binary circulant matrix* A is a $v \times v$ matrix over $\mathbb{F}_2$ defined as

$$A = \begin{bmatrix} a_0 & a_1 & \dots & a_{v-1} \\ a_{v-1} & a_0 & \dots & a_{v-2} \\ \vdots & \vdots & \ddots & \vdots \\ a_1 & a_2 & \dots & a_0 \end{bmatrix},$$

where, for each $0 \leq i < v$, $a_i \in \mathbb{F}_2$. This is by definition a non-singular matrix since every row and column are cyclic shifts of each other.

Definition 2.5.4 (QC Generator Matrix [6, p.24])**.** The Generator matrix G of a quasi-cyclic code has a "block circulant" form

$$G = \begin{bmatrix} G_0 & G_1 & \dots & G_{p-1} \\ G_{p-1} & G_0 & \dots & G_{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ G_1 & G_2 & \dots & G_0 \end{bmatrix},$$

where each block G_i is a matrix of size $k_0 \times n_0$.

This can also be put in a "circulant block form"

$$G = \begin{bmatrix} G^C_{00} & G^C_{01} & \cdots & G^C_{0(n_0-1)} \\ G^C_{10} & G^C_{11} & \cdots & G^C_{1(n_0-1)} \\ \vdots & \vdots & \ddots & \vdots \\ G^C_{(k_0-1)0} & G^C_{(k_0-1)1} & \cdots & G^C_{(k_0-1)(n_0-1)} \end{bmatrix},$$

where each of the blocks $G^C_{i,j}$ is a $p \times p$ binary circulant matrix.

Definition 2.5.5 (QC Parity-Check Matrix [6, p.28])**.** The parity-check matrix H of a quasi-cyclic code has a "block circulant" form

$$H = \begin{bmatrix} H_0 & H_1 & \ldots & H_{p-1} \\ H_{p-1} & H_0 & \ldots & H_{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ H_1 & H_2 & \ldots & H_0 \end{bmatrix},$$

where each block H_i has size $r_0 \times n_0$.

This can also be put in the "circulant block" form

$$H = \begin{bmatrix} H^C_{00} & H^C_{01} & \cdots & H^C_{0(n_0-1)} \\ H^C_{10} & H^C_{11} & \cdots & H^C_{1(n_0-1)} \\ \vdots & \vdots & \ddots & \vdots \\ H^C_{(r_0-1)0} & H^C_{(r_0-1)1} & \cdots & H^C_{(r_0-1)(n_0-1)} \end{bmatrix}.$$

Lemma 2.5.6 ([6, p.33])**.** The set of $v \times v$ binary circulant matrices forms a ring under the standard operations of matrix addition and multiplication in $\mathbb{F}_2$.

Proof. If we consider the algebra of $F[x]/(x^v-1)$, the following map is an isomorphism between such algebra and that of all $v \times v$ circulant matrices over $\mathbb{F}_2$.

$$A \mapsto a(x) = \sum_{i=0}^{v-1} a_i x^i.$$

This associates the first row of the circulant matrix with the polynomial in $\mathbb{F}_2[x]/(x^v - 1)$ given by

$$a(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{v-1} x^{v-1}.$$

The zero element is the all-zero polynomial $a(x) = 0$ and the identity element is $a(x) = 1$. These are associated with the zero matrix and the $v \times v$ identity matrix, respectively. By the rules of polynomial addition and multiplication mod $(x^v - 1)$ the remaining ring properties can be shown. □

The ring of polynomials mod $(x^v - 1)$ contains many polynomials that are zero divisors. The matrices based on these polynomials are not full rank and map to singular circulant matrices which are of lesser utility for error correction as they are not invertible. For example, if there is an even number of nonzero entries in a row, the polynomial is a zero divisor.

Lemma 2.5.7 ([6, p.34])**.** A circulant matrix over $\mathbb{F}_2$ with an even number of nonzero entries in each row (which implies $a(1) = 0$) is rank deficient.

Proof. We observe that each polynomial having the form $1 + x^i,\ 0 \leq i < v$ is a zero divisor in the ring of polynomials (mod $x^v - 1$) over $\mathbb{F}_2$. This is proved by considering that for all $0 \leq i < v$

$$(1 + x^i)(1 + x + x^2 + \cdots + x^{v-1}) \equiv 0 \,(\text{mod } x^v - 1). \tag{2.9}$$

If $\bar{a}(x)$ is a polynomial over $\mathbb{F}_2$ with degree less than v with an even number $2w$ of nonzero coefficients, it can be decomposed into a sum of binomials as follows:

$$\bar{a}(x) = x^{j_1}(1 + x^{i_1}) + x^{j_2}(1 + x^{i_2}) + \cdots + x^{j_w}(1 + x^{i_w}),$$

where $0 \leq j_1, \ldots, j_w < v$, $1 \leq i_1, \ldots, i_w < v$ and $j_l + i_l < v$ for all $1 \leq l \leq w$.

From (2.9) it follows that

$$\begin{aligned}
\bar{a}(1 + x + \cdots + x^{v-1}) = {} & x^{j_1}(1 + x^{i_1})(1 + x + \cdots + x^{v-1}) \\
& + x^{j_2}(1 + x^{i_2})(1 + x + \cdots + x^{v-1}) \\
& + \cdots \\
& + x^{j_w}(1 + x^{i_w})(1 + x + \cdots + x^{v-1}) \\
\equiv {} & 0 \,(\text{mod } x^v - 1).
\end{aligned}$$

□

Theorem 2.5.8 (Invertible Elements in $\mathbb{F}_2/(x^p+1)$ [8, p.12])**.** Let p be a prime number such that $\text{ord}_p(2) = p-1$. Let g be a binary polynomial in $\mathcal{R} = \mathbb{F}_2[x]/(x^p+1)$, with $\deg(g) > 0$. Then g has a multiplicative inverse in $\mathbb{F}_2[x]/(x^p+1)$ if and only if it contains an odd number of terms and $g \neq \Phi$, with $\Phi(x) = x^{p-1} + x^{p-2} + \cdots + x + 1$.

Proof. If $g \in \mathbb{F}_2/(x^p+1)$ has an odd number of terms and $g \neq \Phi$. Let us consider $\gcd(g(x), x^p+1) = \gcd(g, (x+1)\Phi)$. Since $x+1 \nmid g$ as $g(1) = 1$ then $\gcd(g, x+1) = 1$ i.e. they are co-prime. Also, since $\text{ord}_p(2) = p-1$, Φ is irreducible over $\mathbb{F}_2[x]$, which means $g \nmid \Phi$.

Assume, by way of contradiction that $\Phi \mid g$, then $\Phi = gh$ for some $h \in \mathcal{R}$. Then $\deg(g) + \deg(h) = p-1$ but $\deg(g) \leq p-1$. In the case of equality, we would have $\deg(h) = 0$ or $h = 0$ or 1. If $h = 1$ then $gh \mid \Phi$ so $g \cdot 1 \mid \Phi$ which is a contradiction. If $h = 0$ then $g \cdot 0 = \Phi = x^{p-1} + x^{p-2} + \cdots + 1$, which is another contradiction. Hence $\Phi \nmid g$ so $\gcd(g, \Phi) = 1$.

Finally, we know that $\gcd(g(x), x^p+1) = \gcd(g(x), (x+1)\Phi) = 1$ implies that g is invertible.

If $g \in \mathbb{F}_2[x](\ x^p+1)$, with $\deg(g) > 0$ is invertible then

$$\gcd(g(x), x^p+1) = \gcd(g(x), (x+1)\Phi) = 1.$$

Thus,

$$\gcd(g(x), x+1) = 1$$

and

$$\gcd(g, \Phi) = 1$$

implying $g \neq \Phi$ and $g(1) = 1$. Assume, by way of contradiction that there are an even number of terms in g then $g(1) = 0$ which is a contradiction. Hence, the theorem works in both directions. □

Theorem 2.5.9 ([8, p.13])**.** Let $\text{perm}(\cdot)$ be the permanent of a matrix, let $w(\cdot)$ the number of nonzero coefficients of a polynomial (weight), let $p > 2$ be a prime such that $\text{ord}_p(2) = p-1$, and Q be an $n_0 \times n_0$ matrix of elements of $\mathbb{F}_2[x]/(x^p+1)$. If $\text{perm}(w(Q))$ is odd and $\text{perm}(w(Q)) > p$ then Q is non-singular.

Proof. Each block Q_{ij} is isomorphic to a polynomial $q_{ij} \in \mathbb{F}_2[x]/(x^p+1)$, the determinant of the matrix Q can also be represented as an element of $\mathbb{F}_2[x](x^p+1)$. Let $d(x)$ represent the determinant of Q. If the inverse of $d(x)$ exists then Q is non-singular.

Consider two polynomials a, b with $w(a) = w_a$ and $w(b) = w_b$ then

(i) $w(ab) = w_a w_b - 2c_1$, where c_1 is the number of cancellations of pairs of monomials with the same exponent resulting from multiplication.

(ii) $w(a+b) = w_a + w_b - 2c_2$, where c_2 is the number of cancellations due to monomials with the same exponent appearing in both polynomials.

The determinant $d(x)$ is obtained through multiplication and sums of elements q_{ij} and in case of no cancellations has weight equal to perm$(w(Q))$. If some cancellations c_1 or c_2 occur, given (i) and (ii) above, this has no effect on the parity. That is $w(d) =$perm$(w(Q)) - 2c$ or d has odd weight when perm$(w(Q))$ has odd weight. The condition perm$(w(Q)) < p$ guarantees that $d \neq \Phi$ since $w(\Phi) = p$. Since d is odd, and $d \neq \Phi$, by Theorem 2.5.8 Q is non-singular. □

Chapter 3

Public Key Cryptography

3.1 Cryptosystem Definitions

Definition 3.1.1 (Cryptosystem [40, p.345])**.** The general structure of a *cryptosystem* can be described as follows. The main ingredients are:

- An *enciphering scheme* $\mathcal{E}$ for encryption.
- A *deciphering scheme* $\mathcal{D}$ for decryption.
- A key $\mathbf{k}$.
- The *plaintext message* $\mathbf{m}$.
- The *ciphertext* $\mathbf{c}$.

Given a plaintext message $\mathbf{m}$ and a key $\mathbf{k}$, the enciphering scheme $\mathcal{E}_{\mathbf{k}}(\mathbf{m}) = \mathbf{c}$ produces the ciphertext $\mathbf{c}$. The deciphering scheme $\mathcal{D}_{\mathbf{k}}(\mathbf{c}) = \mathbf{m}$ recovers the plaintext message from the ciphertext. If only one key is used, the cryptosystem is a *symmetric key* cryptosystem. An *asymmetric key* or *public key* cryptosystem uses a pair of keys, one for enciphering and a different but related one for deciphering.

3.1.1 Public Key Cryptosystems

The idea of a Public Key Cryptosystem was formally defined in 1976 by Diffie and Hellman. The security of a public key cryptosystem depends upon the concept of a one-way function, a function that is easy to compute but whose inverse is computationally infeasible [23].

Definition 3.1.2 (Computationally Infeasible [23, p.646])**.** A task is *computationally infeasible* if its cost as measured by either the amount of memory used or the runtime is finite but impossibly large.

Definition 3.1.3 (Public Key Cryptosystem [23, p.648])**.** For a keyspace $\mathcal{K}$, a *public key cryptosystem* is a pair of families $\{\mathcal{E}_k\}_{k\in\mathcal{K}}$ and $\{\mathcal{D}_k\}_{k\in\mathcal{K}}$ of algorithms representing invertible transformations,

$$\mathcal{E}_k : \mathcal{M} \to \mathcal{M},$$

$$\mathcal{D}_k : \mathcal{M} \to \mathcal{M},$$

on a finite message space $\mathcal{M}$, such that

1) for every $\mathbf{k} \in \mathcal{K}$, $\mathcal{E}_k$ is the inverse of D_k;

2) for every $\mathbf{k} \in \mathcal{K}$ and $M \in \mathcal{M}$ the algorithms $\mathcal{E}_k$ and $\mathcal{D}_k$ are easy to compute;

3) for almost every $\mathbf{k} \in \mathcal{K}$, each easily computed algorithm equivalent to $\mathcal{D}_k$ is computationally infeasible to derive from $\mathcal{E}_k$;

4) for every $\mathbf{k} \in \mathcal{K}$, it is feasible to compute the inverse pairs $\mathcal{E}_k$ and $\mathcal{D}_k$ from the key $\mathbf{k}$.

Definition 3.1.4 (One-way function [23, p.650])**.** A function f is *one-way* if, for any argument x in the domain of f, it is easy to compute the corresponding value $f(x)$, yet, for almost all y in the range of f, it is computationally infeasible to solve the equation $y = f(x)$ for any suitable argument x.

Definition 3.1.5 (Trap-door function [40, p.347])**.** A function is a *trap-door one-way* function if there is information which when known makes it feasible find an x which solves the equation $y = f(x)$, but without this information f is one-way.

An example of a trap-door function is the discrete log problem. It is easy to compute

$$a \equiv b^x \pmod{p}.$$

if b, x and p are known, but computationally infeasible to solve for b in some groups even if a, x and p are known. Another hard problem of this sort is to factor numbers that are the product of large primes.

One of the first examples of a practical public key cryptosystem was RSA [59], first published in 1978. This system is based on the hard problem of finding factors of n when n is the product of two large primes.

Example 3.1.6 (RSA)**.** The RSA encryption and decryption methods are as follows:

- Select two very large "random" primes p, q.
- Compute $n = p \cdot q$.
- Pick d such that $\gcd(d, (p-1)(q-1)) = 1$.
- Compute e such that $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$.

The public (encryption) key is then (e, n) and the private (decryption) key is (d, p, q). To encrypt a message $\mathbf{m}$ we compute

$$\mathbf{c} = \mathcal{E}(\mathbf{m}) \equiv \mathbf{m}^e \pmod{n}.$$

To decrypt a message $\mathbf{c}$ we compute

$$\mathbf{m} = \mathcal{D}(\mathbf{c}) \equiv \mathbf{c}^d \pmod{n}.$$

The RSA public key cryptosystem is based on the hardness of the problem of finding p and q given only the public key n. One approach to this problem would be to factor $n = p \cdot q$. Thereby making the private key easy to calculate.

In 1994, Shor showed that both the problem of finding discrete logarithms and the problem of factoring large integers could be solved in polynomial time over the length of the input by using the superposition properties of quantum computers [62]. These quantum algorithms reduce the work factor, from exponential to polynomial time, of a number of modern cryptographic schemes including RSA of Example 3.1.6. In fact, the one-way functions underlying the cryptosystem, are no longer computationally infeasible using quantum computers. Thus, prompting the search for one-way functions that do not rely on the factoring large integers or finding the discrete logarithm.

Around the same time RSA was being developed Robert McEliece, in 1978, created a different public key cryptosystem based on coding theory [47].

Example 3.1.7 (McEliece Cryptosystem)**.** The McEliece Cryptosystem encryption and decryption methods are as follows:

- Select n, the number of bits in a message.
- Select t, the number of bits of error correction for a t-error correcting Goppa code.
- Randomly select an irreducible polynomial of degree t over $\mathbb{F}_{2^m}$, where $n \leq 2^m$.
- Produce G, the $k \times n$ generator matrix for the code.
- Select a random dense $k \times k$ non-singular matrix S.
- Select a random $n \times n$ permutation matrix P.
- Compute $G' = SGP$.

The public (encryption) key is then G' and the private (decryption) key is (S, G, P). To encrypt a message $\mathbf{m}$, we randomly select a vector $\mathbf{z} \in \mathbb{F}_2^n$ and compute

$$\mathbf{c} = \mathbf{m}G' \oplus \mathbf{z}.$$

To decrypt a message $\mathbf{c}$, we calculate $\mathbf{c}' = P^{-1}\mathbf{c}$. This produces a codeword in the Goppa code with t-errors. As we show later, these can be corrected using Patterson's algorithm to give $\mathbf{m}' = \mathbf{m}S$. Which can be solved by multiplying by S^{-1} to produce the original message $\mathbf{m}$ [47, p.115].

The McEliece Public Key Cryptosystem is based on the hardness of a general decoding problem for random linear codes which is not currently susceptible to quantum attacks.

A cryptosystem closely related to the McEliece system is that published by Niederreiter in 1986.

Example 3.1.8 (Niederreiter Cryptosystem)**.** The Niederreiter Cryptosystem differs from the McEliece system only in the encryption and decryption algorithms. The public and private keys are generated identically except in this case, we generate a parity-check matrix H instead of a generator matrix G, then proceed to obfuscate it in a similar manner using non-singular matrices M and P.

- Select n, the number of bits in a message.
- Select t, the number of bits of error correction for a t-error correcting Goppa code.

- Randomly select an irreducible polynomial of degree t over $\mathbb{F}_{2^m}$, where $n \leq 2^m$.
- Produce H, the $(n-k) \times n$ parity-check matrix for the code.
- Select a random dense $(n-k) \times (n-k)$ non-singular matrix M.
- Select a random $n \times n$ permutation matrix P.
- Compute $K = MHP$.

To encrypt a message $\mathbf{y} \in \mathbb{F}_q^n$ of weight less than t using the Niederreiter system, we compute

$$\mathbf{c} = K\mathbf{y}^\top.$$

As before, to decrypt a ciphertext $\mathbf{c} = K\mathbf{y}^\top = MHP\mathbf{y}^\top$, we calculate $M^{-1}\mathbf{c} = HP\mathbf{y}$ which can be error corrected to give $P\mathbf{y}$ and then the permutation can be reversed by multiplying by P^{-1} to produce the original message $\mathbf{y}$.

Importantly, prior to the encryption of the message $\mathbf{y}$ can be added to a random codeword from the system and the syndrome decryption will remain identical [51, p.161].

The cryptosystem as proposed by Niederreiter allowed for the choice between Goppa codes or one of several different code families which have since then been broken. For example Reed-Solomon code based cryptosystems were broken in 1992 by Sidelnikov and Chestakov [64].

In fact, several code-based cryptosystem variants have been developed over the years based on the McEliece/Niederreiter skeletons. In order to reduce the size of the public keys for example, some systems use a quasi-cyclic public key in which parts of the larger public key are calculated based on some smaller set of values. This however, in many cases, introduced too much structure to the system so that the message could be easier to extract from just the public keys and an intercepted message.

In the RSA scheme (Example 3.1.6), a given key and plaintext always produces the same ciphertext. By contrast, in the McEliece cryptosystem (Example 3.1.9) a given plaintext can encrypt into different but equivalent ciphertexts. This is due to the fact that the t-errors can be chosen in many different ways. For the original McEliece parameters of $n = 1024, k = 524, t = 50$ there are $\binom{1024}{50}$ ways of doing this and hence, $\binom{1024}{50}$ different possible ways that a plaintext can be encrypted.

3.1.2 Cryptographic System Models

Using public key cryptography as a model, we can build security mechanisms for several different goals. The post-quantum cryptography (PQC) mechanisms under consideration by NIST for standardization [52] have been divided into the following categories:

1. Digital Signature Algorithms, and
2. Public Key Encryption and Key-establishment Algorithms.

Since none of the post-quantum candidates for Digital Signatures uses code based cryptography, the scope of this thesis is limited to Key Establishment Mechanisms (KEM) using Coding Theory as the underlying Public Key Scheme. For more information on other PQC strategies and digital signature proposals, see [52].

A KEM is a cryptosystem whose purpose is to securely negotiate a shared secret key between two systems over an open channel. Some major drawback of public key cryptography are the speed of the algorithms and the communication overhead, that is, more bits are required to be transmitted per message. Due to these drawbacks, public key cryptography is typically used to share a short binary message (on the order of 128 to 512 bits) which can then be used to produce a shared symmetric key. This shared key is then used encrypt larger blocks of data allowing two systems to communicate more quickly and with minimal communication overhead.

3.1.3 Cryptography Threat Models

The Diffie-Hellman paper [23] outlined several possible threats to which a cryptosystem may be subject. Those were a ciphertext only attack, a known plaintext attack and a chosen plaintext attack. In all cases an attacker knows the system parameters and algorithms that comprise the system as these can be determined by inspection of the system and its code.

Definition 3.1.9 (Ciphertext Only Attack [23, p.646])**.** A *ciphertext only attack* is a cryptanalytic attack in which the cryptanalyst possesses only ciphertexts.

Definition 3.1.10 (Known Plaintext Attack [23, p.646])**.** A *known plaintext attack* is a cryptanalytic attack in which the cryptanalyst possesses a substantial quantity of corresponding plaintext and ciphertext.

Definition 3.1.11 (Chosen Plaintext Attack [23, p.646])**.** A *chosen plaintext attack* is a cryptanalytic attack in which the cryptanalyst can submit an unlimited number of plaintext messages of his own choosing and examine the resulting ciphertexts.

The goals and attack models for a cryptosystem are considered separately to build the notion of encryption scheme security. There are formal definitions of all of these terms, but as such security proofs are beyond the scope of this thesis, these are not provided here. The quoted materials [11, 63] contain these formal definitions.

In Bellare, Desai, Pointchval and Rogaway's 1998 paper [11], two different goals *indistinguishability* and *non-malleability* are considered.

Definition 3.1.12 (Indistinguishability [11, p.1])**.** The goal of *indistinguishability (IND)* formalizes an adversary's inability to learn any information about the plaintext x underlying a challenge cipher y.

The goal indistinguishability captures the notion of strong privacy.

Definition 3.1.13 (Non-Malleability [11, p.1])**.** The goal of *non-malleability (NM)* formalizes an adversary's inability, given a challenge ciphertext y, to output a different ciphertext y' such that the plaintexts x, x' underlying these two ciphertexts are "meaningfully related".

The goal of non-malleability captures the notion of strong message integrity under which messages can be considered to be tamper-proof.

The other axis to be considered is that of the attack models. These are, again from Bellare, Desai, Pointchval and Rogaway [11], the models of *chosen plaintext attack (CPA)*, *non-adaptive chosen ciphertext attack (CCA1)* and finally *adaptive chosen ciphertext attack (CCA2).* These are defined as follows.

Definition 3.1.14 (Chosen Plaintext Attack [11, p.1])**.** Under a *chosen plaintext attack (CPA)* the adversary can obtain ciphertexts of plaintexts of her choice. In a public key cryptosystem, having access to the public key suffices.

This modern definition is functionally equivalent to that of Diffie and Hellman (Definition 3.1.11)

Definition 3.1.15 (Non-Adaptive Chosen Ciphertext Attack [11, p.1])**.** Under a *non-adaptive chosen ciphertext attack (CCA1)* the adversary gets, in addition to her abilities under CPA, access to an oracle for the decryption function. This oracle, can only be accessed prior to the attacker obtaining the challenge message y.

Definition 3.1.16 (Adaptive Chosen Ciphertext Attack [11, p.1])**.** In an *adaptive chosen ciphertext attack* (CCA2) the adversary, in addition to its abilities under CCA1, the oracle can now be accessed after obtaining the challenge ciphertext y.

Given the two security goals and three attack models, we can build a table of these notions:

-	CPA	CCA1	CCA2
IND	IND-CPA	IND-CCA1	IND-CCA2
NM	NM-CPA	NM-CCA1	NM-CCA2

Bellare, Desai, Pointchval and Rogaway [11, p.9] also proved implications between several of these notions. Most notable is the fact that IND-CCA2 implies all of the other security models and goals are met. As such, IND-CCA2 is the ultimate goal of modern security schemes.

3.1.4 Fujisaki-Okamoto Conversion

In 1999, Fujisaki and Okamoto proposed a generic scheme combining asymmetric and symmetric cryptographic systems in order to convert a one-way function into a cryptosystem indistinguishable from CCA2 [29]. The general idea is to use a symmetric key cryptosystem to encrypt data and the asymmetric key to pass only enough data to generate the shared secret key and to ensure its integrity.

The Fujisaki-Okamoto conversion requires three asymmetric functions and two symmetric functions. The three asymmetric functions required are

1. *Generate Keypair* Let $\mathcal{K}^a(n_a)$ be a function that takes n_a as a parameter and returns a public and private keypair $((pk, sk) = \mathcal{K}^a(n_a))$, suitable for encrypting the shared secret key $\sigma \in \mathbb{F}_2^{n_s}$ where n_s defines the size of the message space of the symmetric functions (i.e. n_s is the length of the shared secret key used by the symmetric functions).

2. *Encrypt* Let $\mathcal{E}^a_{pk}(\mathbf{m}, \rho)$ be a probabilistic method of encrypting a message $\mathbf{m} \in \mathbb{F}_2^{n_a}$ using the one way function with the public key (pk), and some random data ρ, that is $(\mathbf{c} = (c_1, c_2) = \mathcal{E}^a_{pk}(\mathbf{m}, \rho))$, and

3. *Decrypt* Let $\mathcal{D}^a_{sk}(c_1, c_2)$ be a deterministic method to decrypt the message $\mathbf{m}$ from a ciphertext $\mathbf{c} = c_1, c_2$ given the private key (sk), that is $(\mathbf{m} = \mathcal{D}^a_{sk}(c_1, c_2))$.

The symmetric functions are the encryption function $\mathcal{E}^s_k$, and the decryption function $\mathcal{D}^s_k$, both taking as parameters a message and a shared key. The hash function $\mathcal{G}(\mathbf{m})$, $\mathbf{m} \in \mathbb{F}_2^{n_a}$ is used to convert a message to a symmetric key, and $\mathcal{H}(\sigma, r_1)$ to produce pseudorandom data of length n_a from $\sigma \in \mathbb{F}_2^{n_a}$ and $r_1 \in \mathbb{F}_2^{n_s}$, where n_s defines the message space of the symmetric algorithm.

The Fujisaki-Okamoto conversion consists of the following three functions.

The key generation function is the same as the key generation for the associated asymmetric scheme.

Algorithm 3.1.17 (Fujisaki-Okamoto - Key Generation [29, p.541])**.**

Input: (n_a): The security level, large enough to protect the symmetric key being passed.

Output: (pk, sk): A public/private keypair for the asymmetric function of the appropriate security level to encrypt the shared key.

1: $(pk, sk) \leftarrow \mathcal{K}^a(n_a)$.
2: **return** (pk, sk). =0

The encryption function generates a random seed σ. This σ is then hashed along with the message to create a check value r_1. This ensures that the ciphertext decrypts correctly. Then, σ run through a (possibly different) hash function on its own to generate a shared key r_2. This shared key is encrypted using the asymmetric algorithm to produce c_1. The message is then encrypted using the shared key to produce c_2 and the complete ciphertext (c_1, c_2) is returned.

Algorithm 3.1.18 (Fujisaki-Okamoto - Encryption [29, p.541])**.**

Input: $(\mathbf{m})$: The message as a binary string in $\mathbb{F}_2^n$ (with n defining the size of the message space).

(pk): The public key.

Output: $(\mathbf{c} = c_1, c_2)$: The ciphertext.

1: $\sigma \xleftarrow{\$} \mathbb{F}_2^n$, where n is the length of the message space.
2: $r_1 \leftarrow \mathcal{H}(\sigma, \mathbf{m})$.

3: $r_2 \leftarrow \mathcal{G}(\sigma)$.
4: $c_1 \leftarrow \mathcal{E}^a_{pk}(\sigma, r_1)$, with message σ and r_1 acting as the random string.
5: $c_2 \leftarrow \mathcal{E}^s_{r_2}(\mathbf{m})$, with r_2 as the symmetric key.
6: **return** $\mathbf{c} = (c_1, c_2)$

The decryption algorithm uses the private key sk to decrypt c_1 to produce the seed $\widehat{\sigma}$ and generates the shared secret key $\widehat{r_2}$ by hashing this seed. The shared symmetric key is then used to decrypt the message $\widehat{\mathbf{m}}$ from c_2. Another hash function is run on $\widehat{\sigma}$ and $\mathbf{m}$ to produce the check value $\widehat{r_1}$. If the public key encryption of the message $\mathbf{m}$ with $\widehat{r_1}$ matches c_1 then the message has been decrypted properly, otherwise return failure.

Algorithm 3.1.19 (Fujisaki-Okamoto - Decryption [29, p.541])**.**

Input: $(\mathbf{c} = c_1, c_2)$: The ciphertext.
(sk): The secret key.

Output: $(\mathbf{m})$: The plaintext message or a failure token $\perp$

1: $\widehat{\sigma} \leftarrow \mathcal{D}^a_{sk}(c_1)$ to decrypt the key seed from c_1.
2: $\widehat{r_2} \leftarrow \mathcal{G}(\widehat{\sigma})$ to produce shared symmetric key $\widehat{r_2}$.
3: $\widehat{\mathbf{m}} \leftarrow \mathcal{D}^s_{r_2}(c_2)$. Decrypt the message.
4: $\widehat{r_1} \leftarrow \mathcal{H}(\widehat{\sigma}, \widehat{\mathbf{m}})$. Check that decryption is successful by producing r_1 and using this to verify that it can produce c_1.
5: **if** c_1 is $\mathcal{E}^a_{pk}(\widehat{\sigma}, \widehat{r_1})$ **then**
6: retval $= \widehat{\mathbf{m}}$
7: **else**
8: retval $= \perp$
9: **end if**
10: **return** retval

This Fujisaki-Okamoto conversion has been shown to produce a CCA2 cryptosystem based on an asymmetric cryptosystem with a much weaker security profile, for example, one based only on the hardness of solving a one-way function as in Definition 3.1.4 [29].

3.1.5 Decoding Failure Rate

Depending on the types of cryptosystems used in the Fujisaki-Okamoto conversion it may have nonzero probability of returning a $\perp$ token, or failure for any given

message. For example, one of the first variations on the McEliece cryptosystem using QC-MDPC (Quasi-Cyclic Moderate Density Parity-Check) code decoded with the bit-flipping algorithm was proposed by Misoczki, Tillich and Sendrier [49]. A key-recovery attack on this type of code was produced by Guo, Johansson and Stankovski [35] in which an attacker could eavesdrop and collect the messages which produced a failure and then analyze these to produce the private key. Due to this, it is essential that any cryptosystem based on QC-MDPC has a well-known and negligible decoding error rate.

3.2 The Classic McEliece KEM

The original McEliece PK Cryptosystem (see Example 3.1.9) was designed to be one-way CPA, and does not meet IND-CCA2 security requirements [17, p.5]. The system described here is the modified system by Berstein et al. [17] to meet this modern requirement using the Fujisaki-Okamoto conversion.

3.2.1 Security Base Problem

Definition 3.2.1 ([54, p.9])**.** The *general decoding problem* for linear codes in a norm $\|\cdot\|$ is defined as follows:

- Let $\mathcal{C} \in \mathbb{F}^{k\times n}$ define an (n,k) linear code $\mathcal{C}$ over $\mathbb{F}$ and let $\mathbf{y} \in \mathbb{F}^n$.
- Find $\mathbf{x} \in \mathcal{C}$ where $\|\mathbf{y}-\mathbf{x}\|$ is minimal.

Definition 3.2.2 ([54, p.9])**.** The *problem of finding subspace weights* of a linear code is defined as follows:

- Let $\mathcal{C}$ be an (n,k) linear code over $\mathbb{F}$ and $\mathbf{y} \in \mathbb{F}^n$ with $\|\mathbf{y}\| = w$.
- Find $x \in \mathcal{C}$ where $\|\mathbf{x}\| = w$.

The problem of finding subspace weights of a linear code was originally defined by Berlekamp, McEliece, and Van Tilborg [14] using the Hamming norm.

Theorem 3.2.3 ([54, p.9])**.** The general decoding problem and the problem of finding subspace weights are $\mathcal{NP}$-hard if the norm $\|\cdot\|$ is the Hamming norm [14].

Based on the $\mathcal{NP}$-hardness of the two problems, it is possible to use them to develop a cryptosystem. In 1978, Robert McEliece developed a cryptosystem based on the error correcting Goppa codes.

Definition 3.2.4. [28, p.154] The *McEliece problem* is described as follows: Given a McEliece public key (G, t) where $G \in \mathbb{F}_2^{k \times n}$ and a ciphertext $\mathbf{c} \in \mathbb{F}_2^n$ find the unique message $\mathbf{m} \in \mathbb{F}_2^k$ such that the $d(\mathbf{m}G, \mathbf{c}) = t$.

Any solution to the general decoding problem provides a solution to the McEliece problem, but the reverse may not be true. It has not been proven that the McEliece problem is $\mathcal{NP}$-Hard.

3.2.2 System Parameters

To create a KEM system based on Goppa codes, the following system parameters must be chosen. This is done by weighing security against the size of the public keys and the efficiency of the encrypt/decrypt operations.

Definition 3.2.5 (Classic McEliece (CM) Parameters [17, p.6])**.** A *CM parameter* set specifies the following:

1. A positive integer m. This also defines a parameter $q = 2^m$.
2. A positive integer n with $n \leq q$.
3. A positive integer $t \geq 2$ with $mt < n$. This also defines a parameter $k = n - mt$.
4. A monic irreducible polynomial $f \in \mathbb{F}_2[x]$ of degree m. This defines a representation $\mathbb{F}_2[x]/(f)$ of the field $\mathbb{F}_q$.
5. A positive integer l, and a cryptographic hash function $\mathcal{H}$ that outputs l bits. There are three types of hash inputs: $(2, v)$; $(1, v, C)$; and $(0, v, C)$ with $v \in \mathbb{F}_2^n$, and C the ciphertext. The initial 0, 1 or 2 is used as the first byte of hash data.

3.2.3 Key Generation

In order to generate keys, the following information should be chosen with enough randomness to be cryptographically secure. That is to say, using a true random number generator or, at least a cryptographically secure pseudo-random number generator.

Given a set of CM parameters, a user generates a *CM key pair*.

Algorithm 3.2.6 (Generate CM Key Pair [17, p.8])**.**

Input: (m, n, t, f, l): CM Parameters.

Output: (A): CM Public Key.
(s, Γ): CM Private Key.

1: Generate a uniform random monic irreducible polynomial $g \in \mathbb{F}_q[x]$ of degree t.
2: Select a uniform random sequence $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ of n pairwise distinct elements of $\mathbb{F}_q$ where $g(a_i) \neq 0$.
3: Compute the $t \times n$ matrix $\tilde{H} = \{h_{i,j}\}$ over $\mathbb{F}_q$, where $h_{i,j} = \alpha_j^{i-1}/g(\alpha_j)$ for $i = 1, \ldots, t$ and $j = 1, \ldots, n$.
4: Form an $mt \times n$ matrix $\widehat{H}$ over $\mathbb{F}_2$ by replacing each entry $c_0 + c_1 x + \cdots + c_{m-1}x^{m-1}$ of $\tilde{H}$ with a column of t bits $c_0, c_1, \ldots, c_{m-1}$.
5: Reduce $\widehat{H}$ to systematic form $(I_{n-k} \mid A)$ where I_{n-k} is an $(n-k) \times (n-k)$ identity matrix. If this fails, go back to Step 1.
6: Generate a uniform random n-bit string s.
7: Set $\Gamma = (g, \alpha_1, \alpha_2, \ldots, a_n)$ and output (s, Γ) as private key and A as public key.

The second part of the private key, $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$ describes a binary Goppa code of length n and dimension $k = n - mt$. The public key A is a binary $(n-k) \times k$ matrix such that $H = (I_{n-k} \mid A)$ is a parity-check matrix for the same Goppa code.

The public key A, is the systematic parity-check matrix of a binary Goppa code as opposed to the public key of the original McEliece Cryptosystem (Example 3.1.7), which was the product of three matrices (S, G, P). This allows us to remove $(n-k)^2$ bits from the public key size.

The requirement that a_i is not a root of g is not in the official specification, but is required in order to compute $h_{i,j}$. Otherwise, this produces a failing decapsulation later which prompts restarting the session.

3.2.4 Encoding

The encoding subroutine takes two inputs, a weight-t column vector $\mathbf{e} \in \mathbb{F}_2^n$ and the public key A. The subroutine returns a vector $\mathbf{c}_0 \in \mathbb{F}_2^{n-k}$.

Algorithm 3.2.7 (CM Encoding [17, p.9])**.**

Input: (A): CM Public Key.
$(\mathbf{e})$: A weight-t column vector.

Output: $(\mathbf{c}_0 \in \mathbb{F}_2^{n-k})$: An encoded vector.

1: Define $H = [I_{n-k} \mid A]$.
2: Compute and return $\mathbf{c}_0 = H\mathbf{e} \in \mathbb{F}_2^{n-k}$.

Knowing the form of the parity-check matrix, it is reconstructed from the public key as $[I_{n-k}|A]$ and then used to compute the encode the vector $\mathbf{c}_0 = H\mathbf{e}$.

3.2.5 Decoding

The decoding subroutine takes two inputs, a vector $\mathbf{c}_0 \in \mathbb{F}_2^{n-k}$, and the private key (s, Γ). The subroutine has two possible return values, a weight-t column vector $\mathbf{e} \in \mathbb{F}_2^n$ or a failure token $\perp$.

Algorithm 3.2.8 (CM Decoding [17, p.9])**.**

Input: $(\mathbf{c}_0)$: An encoded vector $\mathbf{c}_0 \in \mathbb{F}_2^{n-k}$.
(s, Γ): the CM Private Key.

Output: $(\mathbf{c}_0 \in \mathbb{F}_2^{n-k})$: An encoded vector.

1: Extend $\mathbf{c}_0$ to $v = (\mathbf{c}_0, 0, \ldots, 0) \in \mathbb{F}_2^n$ by appending k zeros.
2: Find the unique codeword $\mathbf{c}$ in the Goppa code defined by Γ that is at distance at most t from v. If there is no such keyword return $\perp$ (failure).
3: Set $\mathbf{e} = v + c$.
4: **if** $w(\mathbf{e}) = t$ and $\mathbf{c}_0 = H\mathbf{e}$ **then**
5: **return** $\mathbf{e}$.
6: **else**
7: **return** $\perp$.
8: **end if**

In regards to Step 2, there are many different ways of decoding a Goppa code. The Patterson algorithm described in Algorithm 2.3.8 provides an efficient method.

It is observed in [17, p.10] that the triple of algorithms (Key Generation, Encoding, Decoding) is essentially Niederreiter's "dual" version of the McEliece cryptosystem

as in Example 3.1.8 (plus a private string s not used in decoding; s is used in Encapsulation) and without the non-singular S and permutation P matrices which allows us to reduce the size of the public key.

3.2.6 Encapsulation

The encapsulation algorithm is a variation on the Fujisaki-Okamoto conversion in which no message is sent. The goal of the KEM is to generate a shared secret key $k \in \mathbb{F}_2^l$.

The sender generates a session key k and a ciphertext C as follows.

Algorithm 3.2.9 (CM Encapsulation [17, p.10])**.**

Input: (m, n, t, f, l): CM Parameters.
$(\mathcal{H})$: A hash function.

Output: $(\mathbf{k})$: The shared session key.
$(C = \mathbf{c}_0, \mathbf{c}_1)$: The ciphertext.

1: Generate a uniform random vector $\mathbf{e} \in \mathbb{F}_2^n$ of weight t.
2: Use the encoding subroutine on $\mathbf{e}$ and public key T to compute $\mathbf{c}_0$.
3: Compute $\mathbf{c}_1 = \mathcal{H}(2, \mathbf{e})$.
4: Set $C = (\mathbf{c}_0, \mathbf{c}_1)$.
5: Compute $k = \mathcal{H}(1, \mathbf{e}, C)$.
6: **return** session key k and ciphertext C.

The encapsulation approximates the form of the Fujisaki-Okamoto encryption 3.1.18 with no message. A shared secret key is generated on both sides of the channel using the hash of a randomly generated vector $\mathbf{e}$ of weight t.

3.2.7 Decapsulation

The receiver decapsulates the session key $\mathbf{k}$ from ciphertext C as follows.

Algorithm 3.2.10 (CM Decapsulation [17, p.10])**.**

Input: (C): The ciphertext.
(m, n, t, f, l): CM Parameters.
(s, Γ): The private key.
$(\mathcal{H})$: The hash function.

Output: $(\mathbf{k})$: The shared session key.

1: Split the ciphertext C as $(\mathbf{c}_0, \mathbf{c}_1)$ with $\mathbf{c}_0 \in \mathbb{F}_2^{n-k}$ and $\mathbf{c}_1 \in \mathbb{F}_2^l$.
2: Set $b \leftarrow 1$
3: Use the decoding subroutine on $\mathbf{c}_0$ and private key (s, Γ) to compute $\mathbf{e}$. If the subroutine returns $\perp$, set $\mathbf{e} \leftarrow s$ and $b \leftarrow 0$.
4: Compute $\mathbf{c}_1' = \mathcal{H}(2, \mathbf{e})$.
5: If $\mathbf{c}_1' \neq \mathbf{c}_1$, set $\mathbf{e} \leftarrow s$ and $b \leftarrow 0$.
6: Compute $\mathbf{k} = \mathcal{H}(b, \mathbf{e}, C)$.
7: **return** session key $\mathbf{k}$.

If C is a legitimate ciphertext then $C = (\mathbf{c}_0, \mathbf{c}_1)$ with $\mathbf{c}_0 = H\mathbf{e}$ for some $\mathbf{e} \in \mathbb{F}_2^n$ of weight t and $\mathbf{c}_1 = \mathcal{H}(2, \mathbf{e})$. The decoding algorithm returns $\mathbf{e}$ as the unique weight-t vector and the $\mathbf{c}_1' = \mathbf{c}_1$ check passes, thus $b = 1$ and $\mathbf{k}$ matches the session key computed in encapsulation.

If the decoding subroutine returns $\perp$, a session key is still computed in order to meet the random oracle model (CCA2) requirements and to ensure that the algorithms takes a similar amount of time to run even in the event of a failure.

3.2.8 Attacks on System

The Classic McEliece submission to the NIST standardization committee provides an extensive history of attacks on this system from 1981 to 2016. Each of these attacks uses a strategy, developed by Prange in 1962, information-set decoding [57] (ISD). They all, notably, treat the structure of the public key as a uniform random matrix. None of the currently known attacks use the structure of the Goppa codes to retrieve key information. In other words, they attack the general decoding problem and not specifically the McEliece problem.

In Bernstein's 2010 Paper "Grover vs McEliece" [16], the author provides a description of ISD and the use of Grover's quantum searching algorithm in order to attack the McEliece Cryptosystem. The Information Set Decoding algorithm is as follows.

Algorithm 3.2.11 (Information Set Decoding [28, p.172])**.**

Input: (G): A $k \times n$ generator matrix.

$(\mathbf{c} = \mathbf{m}G \oplus \mathbf{e})$: A ciphertext where $\mathbf{m}$ is the plaintext and $\mathbf{e}$ is the error vector of weight t.

(j): A positive integer no bigger than t.

Output: ($\mathbf{m}$): The plaintext.

1: Choose a uniform random size-k subset $S \subseteq \{1, 2, \ldots, n\}$.
2: Select the k columns of G and $\mathbf{c}$ indexed by the set S denote as G_S and $\mathbf{c}_S$, respectively.
3: **if** G_S is invertible **then**
4: Set $Q_1 = G_S^{-1}$ and $Q_2 = Q_1 G$.
5: Set $\mathbf{m}' = \mathbf{c} \oplus \mathbf{c}_S Q_2$.
6: **if** $w(\mathbf{m}') = t$ **then**
7: **return** $\mathbf{m}'$.
8: **end if**
9: **end if**
10: Restart with a new selection for S.

When this algorithm returns, the following two things are true:

- The vector $\mathbf{e}_S = 0$. That is, none of the k values in S indexes a one in the error vector.
- The $k \times k$ matrix G_S is invertible.

Counting, we have the first event occurs $\binom{n-t}{k}$ for each of the $\binom{n}{k}$ possible sets S. On average, this occurs after $\binom{n}{k}/\binom{n-t}{k}$ iterations.

Also, for $R = k/n$ and $t \approx (1-R)n/\log_2(n)$ then

$$\frac{\binom{n}{k}}{\binom{n-t}{k}} = \frac{n \cdots (n-t+1)}{(n-k) \cdots (n-k-t+1)} \approx \left(\frac{n}{n-k}\right)^t = \left(\frac{1}{1-R}\right)^t \approx C^{n/\log_2 n}.$$

where $C = (1/(1-R))^{1-R}$.

Further analysis by Bernstein [16, p.3] shows that

$$\frac{\binom{n}{k}}{\binom{n-t}{k}} \approx {}^{C^{(1+o(1))n/\log_2 n}}$$

when t is sufficiently close to $(1-R)n/\log_2 n$.

In the general case and for McEliece public keys specifically, the matrix G_S is invertible about 29% of the time [16, p.3]. Hence, in the case of the ISD as stated, then success occurs for approximately $0.29\binom{n-t}{k}$ out of $\binom{n}{k}$.

Advanced techniques for producing a smaller number of iterations include allowing e to have a few errors which are further analyzed using combinatorial search techniques, reducing the costs of finding approximate sets S and computing inverses, can speed this ISD by a small constant factor. The most efficient algorithm being a 1988 method of Stern [66]. Regardless, the final cost is still on the order of $c^{(1+o(1))n/\log_2 n}$.

In 1996, Grover developed a search algorithm for quantum computers that reduces the time for searching a database of N records from $O(N)$, on a classical computer, to $O(\sqrt{N})$, on a quantum computer [33].

Grover's quantum search algorithm can be applied to the selection of S, which speeds up the ISD, increasing the probability of success by the square root of the number of classic ISD iterations. Thus, in the case of Quantum ISD, then success requires $\sqrt{\binom{n}{k}/0.29\binom{n-t}{k}} \approx c^{(1/2)n/\log_2 n}$ iterations [16, p.6].

In summary, "With same key-size optimizations, the McEliece system uses a key size of $(c_0 + o(1))b^2(\log(b))^2$ to achieve 2^b bits of security against all non-quantum attacks known today, where c_0 is exactly the same constant. In the quantum case, Grover's algorithm is applicable, reducing the attack cost to its square root. In other words, the key now needs $(4c_0 + o(1))b^2(\log(b))^2$ bits. As before, further papers on the topic have merely improved the $o(1)$ [17, p.16]."

3.2.9 Security Analysis

Some of the recommended and implemented sets of parameters are as follows.

ID	m	n	t	l	field polynomial
mceliece348864	12	3488	64	256	$f(x) = z^{12} + z^3 + 1$
mceliece460896	13	4608	96	256	$f(x) = z^{13} + z^4 + z^3 + z + 1$
mceliece6688128	13	6688	128	256	$f(x) = z^{13} + z^4 + z^3 + z + 1$

For example, in mceliece34886, we have $m = 12$, $n = 3488 \leq 2^m$, $t = 64$ and $l = 256$. The number of possible vectors of weight t in an n length message is $\binom{n}{t} = \binom{3488}{64} \approx 2.3 \times 10^{137}$, which is greater than the number of possible shared keys $k = 2^{256} \approx 1.2 \times 10^{77}$. The hash function in the recommended implementation

is SHAKE256, which produces a pseudo-random 256-bit value from hashing one of these $\binom{3488}{64}$ error vectors.

In the event of a failure during decoding a hash is performed on some "garbage" data, this prevents timing and power based side-channel attacks since the same operations take place even in the event of a KEM failure.

3.3 BIKE

The Bit-Flipping Key Encapsulation (BIKE) standard [5] is an IND-CCA Key Encapsulation Mechanism (KEM) using QC-MDPC for creation, encapsulation and decapsulation of a 256-byte shared secret key.

3.3.1 Security Base Problem

The hard problems upon which the security of BIKE is based are analogous to those of Definitions 3.2.1 and 3.2.2 using a Hamming norm. Both of these have been shown to be $\mathcal{NP}$-complete by Berlekamp, McEliece and Van Tilborg in [14] for generic (non-QC) codes.

Definition 3.3.1 (Problem 1 - Syndrome Decoding (SD) [3, p.48])**.**
The *Syndrome Decoding (SD) problem* for QC-MDPC codes is defined as follows:

- Let $\mathcal{C}$ be an (n, k) t-error correcting linear code over $\mathbb{F}$, let $H \in \mathbb{F}_2^{(n-k)\times n}$ be its parity-check matrix, and let $s \in \mathbb{F}_2^n$.
- Find $\mathbf{e} \in \mathbb{F}_2^n$ such that $|\mathbf{e}| \leq t$ and $\mathbf{e}H^T = s$.

In the generic (non-QC) case, this is analogous to the problem of finding weights described in Definition 3.2.2.

Definition 3.3.2 (Problem 2 - Codeword Finding (CF) [3, p.49])**.**
The *Codeword Finding (CF) problem* for QC-MDPC codes is defined as follows:

- Let $\mathcal{C}$ be an (n, k) t-error correcting linear code over $\mathbb{F}$, and let $H \in \mathbb{F}_2^{(n-k)\times n}$ be its parity-check matrix.
- Find $\mathbf{c} \in \mathbb{F}_2^n$ such that $|\mathbf{c}| = t$ and $\mathbf{c}H^T = 0$.

In the generic case, this is analogous to the general decoding problem of Definition 3.2.1.

For the following examples and definitions, $\mathcal{R}$ represents the cyclic polynomial ring $\mathbb{F}_2[X]/(X^r - 1)$ and $|x|$ represents the number of nonzero coefficients of the binary polynomial x.

A (2,1)-QC parity-check matrix H over $\mathcal{R}$ is in systematic form if it can be written as $\begin{pmatrix} 1 & h \end{pmatrix}$ with $h \in \mathcal{R}$.

For the quasi-cyclic (QC) cases, related problems are believed (not proven) to be $\mathcal{NP}$-hard. These associated problems are the following.

Definition 3.3.3 (Problem 3 - $(2,1)$-QC Syndrome Decoding - $(2,1)$-QCSD [5, p.18])**.** The $(2,1)$-*QC Syndrome Decoding (QCSD) Problem* for QC-MDPC codes is defined as follows:

- Let $\mathcal{C}$ be an (n,k) t-error correcting linear code over $\mathcal{R}$ and given $s, h \in \mathcal{R}$.
- Find $\mathbf{e}_0, \mathbf{e}_1 \in \mathcal{R}$ such that $|\mathbf{e}_0| + |\mathbf{e}_1| \leq t$ and $\mathbf{e}_0 + \mathbf{e}_1 h = s$.

The message security notion (the non-malleability of Definition 3.1.13) captured by the BIKE-KEM relies on a decisional variant of Problem 3 in which an attack must be able to decide, for an appropriate (s, h) whether an error vector exists which matches. The main differences being parity-checks on $w(h)$ and $w(c)$ and the equality of the error weight. By construction, $w(h)$ is always odd as the weight of a public keys is odd. Problems 3 and 3a below are of similar difficulty whether or not equality is used.

Definition 3.3.4 (Problem 3a - $(2,1)$-QC Decisional Parity - $(2,1)$-QCSD [5, p.18])**.** The $(2,1)$-*QC Decisional Parity (QCSD) Problem* for QC-MDPC codes is defined as follows:

- Let $\mathcal{C}$ be an (n,k) t-error correcting linear code over $\mathcal{R}$ and given $s, h \in \mathcal{R}$, with $w(h)$ odd and $w(\mathbf{c}) + t$ even.
- Is there a $\mathbf{e}_0, \mathbf{e}_1 \in \mathcal{R}$ such that $|\mathbf{e}_0| + |\mathbf{e}_1| = t$ and $\mathbf{e}_0 + \mathbf{e}_1 h = \mathbf{c}$?

Definition 3.3.5 (Problem 4 - $(2,1)$-QC Codeword Finding - $(2,1)$-QCCF [5, p.18])**.** The $(2,1)$-*QC Codeword Finding (QCCF)* for QC-MDCP codes is defined as follows:

- Let $\mathcal{C}$ be an (n, k) t-error correcting linear code over $\mathcal{R}$ and given $h \in \mathcal{R}$.
- Find $\mathbf{c}_0, \mathbf{c}_1 \in \mathcal{R}$ such that $|\mathbf{c}_0| + |\mathbf{c}_1| = t$ and $\mathbf{c}_0 + \mathbf{c}_1 h = 0$.

The indistinguishability notion (IND of Definition 3.1.12) is captured by a variation of Problem 4.

Definition 3.3.6 (Problem 4a - $(2, 1)$-QC Decisional Balanced - $(2, 1)$-QCCF [5, p.19])**.** The $(2, 1)$-*QC Decisional Balanced Problem (QCCF)* for QC-MDCP codes is defined as follows:

- Let $\mathcal{C}$ be an (n, k) t-error correcting linear code over $\mathcal{R}$ and given $h \in \mathcal{R}$ with $w(h)$ odd and an integer $w > 0$ such that w is even and $w/2$ is odd.
- Is there a $\mathbf{h}_0, \mathbf{h}_1 \in \mathcal{R}$ such that $|\mathbf{h}_0| + |\mathbf{h}_1| = w/2$ and $\mathbf{h}_0 + \mathbf{h}_1 h = 0$?

According to the authors of the specification, none of the variations described have an impact of the hardness of the problems [5, p.19].

3.3.2 System Parameters

The selection of system parameters is based upon setting a security level λ and then using λ to generate the system.

Algorithm 3.3.7 (BIKE-Parameter Setup [5, p.3])**.**

Input: (λ): The target security level.

Output: (r): The block length.
(w): The row weight(w).
(t): The decoding radius.
(l): The size of the shared secret.
$(\mathcal{H}, \mathcal{K}, \mathcal{L})$: A set of hash functions also required for use in the KEM).

1: Given λ, set the paramters r, w, t and l from the recommended values. Set l to 256.
2: **if** λ is 1 **then**
3: Set $r = 12,323, w = 142$, and $t = 134$.
4: **end if**
5: **if** λ is 3 **then**
6: Set $r = 24,659, w = 206$, and $t = 199$.

7: **else**
8: End ▷ λ is undefined.
9: **end if**
10: Select a hash function $\mathcal{H} : \mathbb{F}_2^l \to \mathbb{F}_2^{2r}$ which returns vector of weight t and length $n = 2r$.
11: Select a hash function $\mathcal{K} : \mathbb{F}_2^{r+2l} \to \mathbb{F}_2^l$.
12: Select a hash function $\mathcal{L} : \mathbb{F}_2^{2r} \to \mathbb{F}_2^l$.
13: **return** $\mathcal{H}, \mathcal{K}, \mathcal{L}, r, w, t, l$

This inherently defines $n = 2r$ and $d = w/2$.

The function $\mathcal{H}$ uses the 256-bit AES in counter mode (AES256-CTR) to expand a given seed of length l into a pseudo-random string of bits with a length $2r$ of weight t. The functions $\mathcal{K}$ and $\mathcal{L}$ use standard SHA384 hash functions discarding the bits above l bits, collapsing the output to 256 bits.

BIKE targets security levels 1 and 3 from the NIST call for proposals. This is basically defined as the security of AES-128 and AES-192, respectively. For all security levels, the size of the shared secret key (l) is set to 256. The other values are given in the suggested BIKE parameters table from the specification and repeated here [5, p.9].

Security	r	w	t	DFR
Level 1	12,323	142	134	2^{-128}
Level 3	24,659	206	199	2^{-192}

The algorithms described next require functions to uniformly and randomly generate vectors and keys, sometimes of specific weights. This is denoted by $(u \xleftarrow{\$} U)$.

3.3.3 Key Generation

Algorithm 3.3.8 (BIKE Key Generation [3, p.4])**.**

Input: (n, w, t, l): BIKE Parameters.

Output: (h_0, h_1, σ): The private key.
(h): The public key.

1: Generate $h_0, h_1 \xleftarrow{\$} \mathcal{R}^2$ both of odd weight $|h_0| = |h_1| = w/2$.

2: Generate $\sigma \xleftarrow{\$} \mathbb{F}_2^l$ uniformly at random.
3: Compute $h \leftarrow h_1 h_0^{-1}$.
4: **return** (h_0, h_1, σ) and h.

One of the major challenges in BIKE Key Generation is to produce h_0 such that it is invertible. The algorithm for generation of invertible polynomials in BIKE is based on the 2020 work of Drucker, Gueron and Kostic [25]. In this paper the authors generalize the fast k-squaring algorithm of Itoh and Tsujii (ITI algorithm) [38] to show that raising an element a to the power of 2^k (or k-squaring) can be done efficiently in the polynomial ring $\mathcal{R} = \mathbb{F}_2[x]/(x^r - 1)$ as well.

Algorithm 3.3.9 (Computing a^{2^k-1} where $k = 2^t$ (ITI) [25, p.4])**.**

Input: a

Output: a^{2^k-1}

1: $f = a$
2: **for** $i = 0$ **to** $t - 1$ **do**
3: $\quad g = f^{2^{2^i}}$
4: $\quad f = f \cdot g$
5: **end for**
6: **return** f

The inverse of $a \in \mathbb{F}_{2^l}$ for $l = 2^t + 1$ can be computed based on Fermat's Little Theorem as

$$a^{-1} = a^{2^l-2} = (a^{2^{l-1}-1})^2 = (a^{2^{2^t}-1})^2. \tag{3.1}$$

Operating in $\mathcal{R} = \mathbb{F}_2[x]/(x^r - 1)$ for some r, since $\text{ord}(a) \mid 2^{r-1} - 1$ for every $a \in \mathcal{R}^*$, the inverse can be shown to be

$$a^{-1} = a^{2^{r-1}-2} = (a^{2^{r-2}-1})^2.$$

Because $r - 2$ is not a power of 2 we cannot directly use (3.1) to calculate $(a^{2^{r-2}-1})^2$. The following decomposition is used:

$$s = \text{supp}(r-2) \quad \text{(that is, the position of the set bits of } r-2\text{)},$$
$$z = 2 \cdot (2^{r-2} - 1) = 2 \cdot \sum_{i \in s} \left((2^{2^t} - 2) \cdot \left(2^{(r-2) (\text{mod } 2^t)} \right) \right).$$

Algorithm 3.3.10 (Inversion of a in $\mathcal{R}^*$ [25, p.4])**.**

Input: $a \in \mathcal{R}^*$.

Output: a^{-1}.

1: $f = a$
2: $res = a$
3: **for** $i = 0$ to $\lfloor \log(r-2) \rfloor$ **do**
4: $\quad g = f^{2^{2^{(t-1)}}}$
5: $\quad f = f \cdot g$
6: $\quad$ **if** $((r-2)[i] = 1)$ (where $(r-2)[i]$ is the ith bit of the binary expansion of $r-2$) **then**
7: $\quad\quad res = res \cdot f^{2^{(r-2) \pmod{2^i}}}$
8: $\quad$ **end if**
9: **end for**
10: $res = res^2$
11: **return** res

The value of r is carefully chosen so that the weight of $r-2$ is small. For example, the values for r above are 12,323 and 24,659. The values of $w(12,321) = 4$ and $w(24,657) = 5$. In order to reduce timing based information leaks, the inversion algorithm is implemented in constant time [18].

3.3.4 Encapsulate

Algorithm 3.3.11 (BIKE Key Encapsulation [3, p.4])**.**

Input: (h): The public key.

Output: $(\mathbf{k})$: The encapsulated key.
$(C = (c_0, c_1))$: The ciphertext.

1: Generate $\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^l$ uniformly at random.
2: Compute $(e_0, e_1) \leftarrow \mathcal{H}(\mathbf{m})$.
3: Compute $(c_0, c_1) \leftarrow (e_0 + e_1 h, \mathbf{m} \oplus \mathcal{L}(e_0, e_1))$.
4: Compute $\mathbf{k} \leftarrow \mathcal{K}(\mathbf{m}, C)$.
5: **return** (C, K)

The encapsulate function follows the form of the Fujisaki-Okamoto conversion of Algorithm 3.1.18 with no associated message in order to pass the shared secret key.

In step (1) we generate this secret key. This secret key is used to produce an error vectors e_0, and e_1, which are encoded using the public key to produce the first half of the ciphertext $c_0 = e_0 + e_1 h$. The second half hashes the entire error vector (e_0, e_1) and XORs this with the message to produce c_1. The secret key is then calculated by hashing $K = \mathcal{K}(\mathbf{m}, C)$.

3.3.5 Decapsulate

Algorithm 3.3.12 (BIKE Key Decapsulation [3, p.14])**.**

Input: (h_0, h_1, σ): The private key.
$(C = (c_0, c_1))$: The ciphertext.

Output: $(\mathbf{k})$: The decapsulated key.

1: Generate $(e'_0, e'_1) \xleftarrow{\$} \mathcal{R}^2$.
2: Compute the syndrome $s \leftarrow c_0 h_0$.
3: Compute $\{(e''_0, e''_1), \perp\} \leftarrow decoder(s, h_0, h_1)$
4: **if** $(e''_0, e''_1) \leftarrow \text{decoder}(s, h_0, h_1)$ and $|(e''_0, e''_1)| = t$ **then**
5: Set $(e'_0, e'_1) \leftarrow (e''_0, e''_1)$.
6: **end if**
7: $\mathbf{m}' \leftarrow c_1 \oplus \mathcal{L}(e'_0, e'_1)$.
8: **if** $\mathcal{H}(\mathbf{m}') \neq (e'_0, e'_1)$ **then**
9: Compute $\mathbf{k} \leftarrow \mathcal{K}(\sigma, C)$.
10: **else**
11: Compute $\mathbf{k} \leftarrow \mathcal{K}(\mathbf{m}', C)$.
12: **end if**
13: **return k**.

The decapsulation algorithm is based on the Fujisaki-Okamoto conversion Algorithm 3.1.19. The calculation at (6) retrieves the initial message $\mathbf{m}$ which was encoded in Encapsulation step (3). We know that $\mathcal{H}(\mathbf{m}) = (e_0, e_1)$ from Encapsulation, so if, at step (7) these are not equal then we will compute some garbage data in order that information does not leak via timing differences. In the case we reach step (10), we have a valid key.

The algorithm is dependent on an associated decoder function that returns the error vector (e_0, e_1) or an error token $(\perp)$. The probability of producing this error token during a key exchange session is the *Decoding Failure Rate (DFR)* and this

can be seen in the parameter table. It is dependent only on the algorithm chosen for the decoder function. This allows for future improvement to the decoder, either for speed, security or to reduce failures without affecting the outlying security scheme.

The decoder function currently recommended for BIKE is the Black-Gray-Flip decoder of Algorithm 3.3.13, an improvement by Drucker, Gueron and Kostick [26] on the bit-flipping algorithm originally invented in 1962 by Gallager [30]. Drucker, Gueron and Kostic show that this decoder has a failure rate of less than $1/2^{-128}$ in [24] which is what is required in order to be IND-CCA. This is a constant time algorithm to avoid information leaking through timing differences.

The Black-Gray-Flip decoder requires threshold and ctr functions. The threshold(S, i) function is for threshold selection. In general it depends on the syndrome S, and the current iteration number i. The function in the BIKE specification is independent of i and is currently a parameter of the security level λ. It is defined for both Security level $\lambda = 1$, or $\lambda = 3$ [5, p.5-6]. For security level 1,

$$\text{threshold}(S, i) = \max(\lceil 0.0069722 \cdot S \rceil + 13.530, 36).$$

For security level 3,

$$\text{threshold}(S, i) = \max(\lceil 0.005265 \cdot S+ \rceil + 15.2588, 52).$$

These threshold values are based on the PhD thesis of Chaulet [22], who showed that the best threshold value is the smallest integer T such that for

$$\pi_1 = \frac{|s| + X}{td}, \quad \text{and} \quad \pi_0 = \frac{w|s| - X}{(n-t)d},$$

where

$$X = \sum_{l_{odd}} (l-1) \frac{r\binom{w}{l}\binom{n-w}{t-l}}{\binom{n}{t}},$$

then

$$T = \left\lceil \frac{\log \frac{n-t}{t} + d \log \frac{1-\pi_0}{1-\pi_1}}{\log \frac{\pi_1}{\pi_0} + \log \frac{1-\pi_0}{1-\pi_1}} \right\rceil,$$

which depends on the values $n = 2r$, w and t defined for a security level $\lambda = 1$ or $\lambda = 3$ and S, the syndrome weight [4]. The ctr(H, s, j) function computes a count of the number of unsatisfied parity-checks of j. It is the number of bits that appear in

the same position in the syndrome s and the jth column of the matrix H.

The Black-Gray-Flip decoder described below relies on the two procedures BFIter(s, e, T, H) and BFMaskedIter$(s, e,$ mask, $T, H)$. The BFIter computes one iteration of Bit-Flipping followed by setting the black and gray mask array. Then in BFMaskedIter, the black and gray masks are used in performing a second and third iteration of Bit-Flipping in which the error e is only updated based on the content of these masks. The result of this is that the Black-Gray-Flip needs only 7 steps to an acceptable DFR (i.e. less than $1/2^{-128}$) and a DFR lower than the comparable Black-Gray decoder of Drucker, Gueron, and Kostic [24] can achieve in 9 steps [26].

Algorithm 3.3.13 (Black-Gray-Flip (BGF) [5, p.6])**.**

Input: ($\mathbf{s} \in \mathbb{F}_2^r$): The received message.
($H \in \mathbb{F}_2^{r \times n}$): The parity-check matrix.
$(r, w, t, d = w/2, n = 2r)$: The BIKE parameters.
(NbIter): The number of iterations.
(τ): A threshold value.

Output: (**e**):The recovered error vector.
($\perp$): A possible error token if the error cannot be recovered.

procedure BFIter(s, e, T, H)
  for $j = 0$ **to** $n - 1$ **do**
    if $\mathrm{ctr}(H, s, j) \geq T$ **then**
      $e_j \leftarrow e_j \oplus 1$
      $\mathrm{black}_j \leftarrow 1$
    else if $\mathrm{ctr}(H, s, j) \geq T - \tau$ **then**
      $\mathrm{gray}_j \leftarrow 1$
    end if
  end for
  return e, black, gray
end procedure
procedure BFMaskedIter$(s, e, \mathrm{mask}, T, H)$
  for $j = 0$ **to** $n - 1$ **do**
    if $\mathrm{ctr}(H, s, j) \geq T$ **then**
      $e_j \leftarrow e_j \oplus \mathrm{mask}_j$
    end if
  end for

return e
end procedure
Set $\mathbf{e} \leftarrow \{0\}^n$
for $i = 0, \ldots,$ NbIter **do**
    $T \leftarrow$threshold$(|s + eH^T|, i)$
    e, black, gray $\leftarrow$ BFIter$(s + eH^T, e, T, H)$
    if i=1 **then**
        $e \leftarrow$ BFMaskedIter$(s + eH^T, e,$ black$, (d+1)/2+1, H)$
        $e \leftarrow$ BFMaskedIter$(s + eH^T, e,$ gray$, (d+1)/2+1, H)$
    end if
end for
if $s = eH^T$ **then**
    return e
else
    return $\perp$
end if

3.3.6 Attacks on System

As in the case of the McEliece Cryptosystem, the best attack on the BIKE system is a variant of Prange's ISD [57]. Let $WF_{\mathcal{A}}(n, k, t)$ be the amount of work required by the best ISD variant $\mathcal{A}$ which decodes a length n, dimension k, t-error correcting binary code. Then the work factor can be written as

$$WF_{\mathcal{A}}(n, k, t) = 2^{ct(1+o(1))},$$

where c depends on the algorithm, the code rate $R = k/n$ and the error rate t/N. Torres and Sendrier in [70] prove that for weight $t = o(n)$, specifically where $w \approx t \approx \sqrt{n}$, we have $c = \log_2 \frac{1}{1-R}$ which implies $WF_{\mathcal{A}}(n, k, t) \approx 2^{l \log_2 \frac{n}{n-k}}$ for all ISD variants [5, p.20].

Due to the quasi-cyclic nature of BIKE, it is easier to perform codeword finding and syndrome decoding by a factor of r. To reach $\lambda \in 128, 256$ bits of classical security, with $WF_{\mathcal{A}}(n, k, t) = 2^{l \log_2 \frac{n}{n-k}}$, we choose w, t, and r such that

$$\lambda \approx t - \frac{1}{2} \log_2 r \approx w - \log_2 r,$$

in order that problems 3a and 4a of Definitions 3.3.4 and 3.3.6 be "hard enough" [5, p.21]. It is also required that r has the following properties.

- 2 is primitive modulo r, and
- r is prime.

This choice thwarts the squaring attack of [43] as well as avoids attempts to exploit, by factoring, the structure of $\mathbb{F}_2[x]/(x^r - 1)$ by ensuring that $x^r - 1$ has only two factors (one of which is $x - 1$).

As discussed during the analysis of the Classic McEliece system, any known quantum speedup is based on Grover's search algorithm applied to ISD and is at best quadratic [5, p.22].

3.3.7 Security Analysis

These limitations of the values of r, w, and t lead to the results that were in parameter setup of Algorithm 3.3.7.

Security	r	w	t	DFR
Level 1	12,323	142	134	2^{-128}
Level 3	24,659	206	199	2^{-192}

Another consideration, in addition to the security limitations, was that the block length r is chosen so that the $|r - 2|$ (the Hamming weight of the binary expansion of $r - 2$) is small in order that the inversion algorithm (Algorithm 3.3.10) [25] be as efficient as possible. For $r = 12,323$, $|r - 2| = 4$ and for $r = 24659$. $|r - 2| = 5$.

3.4 LedaCrypt

LEDAcrypt (Low-dEnsity parity-check coDe-bAsed cryptographic systems) is another cryptosystem proposal based on low-density quasi-cyclic (QC-LDPC) codes. It consists of 3 different cryptographic primitives [8, p.8]:

1. LEDAcrypt-KEM, an IND-CCA2 key encapsulation method,

2. LEDAcrypt-PKC, an IND-CCA2 public key encryption scheme and,

3. LEDAcrypt-KEM-CPA, an IND-CPA key encapsulation method which has been optimized for use in an ephemeral key scenario.

We consider the LEDAcrypt-KEM, which is comparable to the KEMs in BIKE and Classic McEliece.

As in the case of BIKE, LEDAcrypt uses quasi-cyclic codes over the ring $\mathcal{R} = \mathbb{F}_2[x]/(x^v + 1)$ represented with a binary circulant matrix as in Definition 2.5.3 with a polynomial analogue as in Definition 2.5.6.

3.4.1 Security Base Problem

Like BIKE above, the hard problems on which the security of LEDAcrypt is based are the syndrome decoding problem of Definition 3.3.1 and the general decoding problem described above in Definition 3.3.2 which were shown to be $\mathcal{NP}$-complete in [14] for the general case, and assumed to be at least $\mathcal{NP}$-hard for the quasi-cyclic case.

The fast decoding algorithm of the QC-LDPC codes in LEDAcrypt is a flavour of the Bit-Flipping Decoding algorithm customized for sparse parity-check matrices. In this algorithm, the BITTOGGLE() function toggles the bit referred to in the argument. The H_{sparse} matrix is a $d_v \times n_0$ integer matrix containing in each of its n_0 columns, the position in $\{0, 1, \ldots, pr_0 - 1\}$ of the asserted binary coefficients in the first column of each circulant block in the sequence of n_0 submatrices in $H = [H_0|H_1|\ldots|H_{n_0-1}]$ (any of which with size $p \times p$).

Algorithm 3.4.1 (LEDAcrypt Bit-Flipping Decoding [9, p.18])**.**

Input: (x): The received QC-LDPC codeword, with errors. (A $1 \times pn_0$ binary vector.)

(s): The QC-LDPC syndrome ($s = Hx^T$). (A $pr_0 \times 1$ binary vector.)

(H_{sparse}): A sparse version of the parity-check matrix $H_{pr_0 \times pn_0}$. (A $d_v \times n_0$ integer matrix.)

(imax): The maximum number of iterations before reporting a decoding error.

Output: (c): The error-free $1 \times pn_0$ codeword.

(decodeOk): A Boolean value, true if outcome of decoding is successful.

1: codeword $\leftarrow x$ (bitvector with size pn_0)

2: syndrome $\leftarrow s$ (bitvector with size pr_0)
3: iter $\leftarrow 0$ (scalar variable denoting the number of iterations)
4: **repeat**
5: iter $\leftarrow$ iter $+ 1$
6: unsatParityChecks $\leftarrow 0_{1\times pr_0}$ (counters of unsatisfied parity-checks)
7: **for** $i = 0$ **to** n_0 **do**
8: **for** exp $= 0$ **to** p_1 **do**
9: **for** $j = 0$ **to** $d_v - 1$ **do**
10: assertedHbitPos $\leftarrow$
$(\text{exp}+H_{sparse}[j][i]) \mod p + p \cdot \lfloor H_{sparse}[j][i] \text{div } p \rfloor$
11: **if** syndrome[assertedHbitPos] $= 1$ **then**
12: unsatParityChecks[$i \cdot p$+exp] $\leftarrow$ 1+unsatParityChecks[$i \cdot p$+exp]
13: **end if**
14: **end for**
15: **end for**
16: **end for**
17: threshold $\leftarrow$ thresholdChoice(iterationCounter, syndrome)
18: **for** $i = 0$ to $pn_0 - 1$ **do**
19: **if** unsatParityChecks[i] $\geq$ threshold **then**
20: BITTOGGLE(codeword[i]) (update the codeword)
21: **for** $j = 0$ to $d_v - 1$ **do**
22: assertedHbitPos $\leftarrow$
23: $(\text{exp}+H_{sparse}[j][i]) \mod p + p \cdot \lfloor H_{sparse}[j][i] \text{div } p \rfloor$
24: BITTOGGLE(syndrome[assertedHbitPos])
25: **end for**
26: **end if**
27: **end for**
28: **until** syndrome $\neq 0_{1\times pr_0}$ AND iter $<$ imax
29: **if** syndrome $= 0_{1\times pr_0}$ **then**
30: **return** codeword, true
31: **else**
32: **return** codeword, false
33: **end if**

This is a fairly standard bit-flipping implementation on a packed sparse LDPC

check array. First the number of unsatisfied parity-checks are calculated for each bit of the received word, and the maximum number of unsatisfied checks is calculated. Then a decision is made to flip all of the bits whose number of unsatisfied checks matches this maximum number (this flips at least one bit). This process is repeated until the maximum number of iterations is reached or the syndrome of the updated codeword is zero.

Several different types of decoders have been developed for QC-LDPC purposes based on this BF-decoder. One iteration of the BF-decoder, that is lines 4-19 of Algorithm 3.4.1 begins with an estimation of the value of the syndrome (or associated e) then calculates the number of unsatisfied error checks and produces and new, closer, value of the syndrome by toggling the the number of unsatisfied parity-checks. One pass through this process is known as an *iteration* of the BF-decoder.

LEDAcrypt mainly uses two different flavors of iterations in the decoders known as *in-place* iterations and *out-of-place iterations*. The main difference between the two algorithms lies in when the update of the syndrome occurs. In an in-place iteration, the syndrome is updated after determining if the j-th bit is to be flipped or not, and in an out-of-place iteration, the syndrome is updated after the flip in the estimated error vector e.

The two algorithms are as follows.

The randomized in place iteration determines first a permutation which provides the order in which the n iterations of the loop will loop at the columns of the parity-check matrix.

Algorithm 3.4.2 (Randomized In Place Iteration [9, p.52])**.**

Input: (s): QC-LDPC syndrome, binary vector of size p.

(Htr): The transposed parity-check matrix, represented as an $n_0 \times v$ integer matrix $H = [H_0|H_1|\dots|H_{n_0-1}]$.

$(\widehat{e})$: The initial error vector estimate, binary vector of size $n = n_0 p$.

$(\text{iter}^{\text{in}})$ The number of iterations computed.

Output: $(\bar{e})$: updated error vector estimate.

(s): updated syndrome.

1: $\pi \xleftarrow{\$} S_n$ ▷ Random permutation of size $n = n_0 p$.

2: $J \leftarrow \pi(\langle 0, 1, , \dots, n-1 \rangle)$

▷ Permutation of the sequence $\langle 0, 1, \dots, n-1 \rangle$.

3: th $\leftarrow$ ComputeThreshold(iter^{In})

4: **for** $j \in J$ **do**
5: $i \leftarrow \lfloor j/p \rfloor$ ▷ Set circulant block index.
6: offset $\leftarrow j \pmod p$ ▷ Set offset.
7: $\text{upc}_j \leftarrow 0$ ▷ Initialize count of unsatisfied parity-checks.
8: **for** $k = 0$ **to** $v - 1$ **do**
9: **if** GetSyndromeBits$(s, \text{Htr}[i][k] + \text{offset}) \pmod p = 1$ **then**
10: $\text{upc}_j \leftarrow \text{upc}_j + 1$ ▷ Count unsatisfied parity-checks in column j.
11: **end if**
12: **end for**
13: **if** $\text{upc}_j \geq \text{th}$ **then**
14: $\bar{e_j} \leftarrow \widehat{e_j} \oplus 1$ ▷ Update the estimated error, if the number of unsatisfied parity-checks is larger than the threshold value.
15: **for** $k = 0$ **to** $v - 1$ **do**
16: $\text{idx} \leftarrow (\text{Htr}[i][k] + \text{offset}) \pmod p$
17: $s_{\text{idx}} \leftarrow s_{\text{idx}} \oplus 1$ ▷ Update the syndrome with the new value of $\bar{e_j}$.
18: **end for**
19: **else**
20: $\bar{e_j} \leftarrow \widehat{e_j}$ ▷ Keep the error estimate the same at that index.
21: **end if**
22: **end for**
23: **return** $\{\bar{e}, s\}$

In the out of place iteration, the update to the syndrome takes place after the update to the error vector **e**.

Algorithm 3.4.3 (Out Of Place Iteration [9, p.59])**.**

Input: (s): QC-LDPC syndrome, binary vector of size p.
(Htr): The transposed parity-check matrix, represented as an $n_0 \times v$ integer matrix $H = [H_0|H_1|\dots|H_{n_0-1}]$.
$(\widehat{e})$: The initial error vector estimate, binary vector of size $n = n_0 p$.
$(\text{iter}^{\text{in}})$ The number of iterations computed.

Output: $(\bar{e})$: updated error vector estimate.
(s): updated syndrome.

1: currSynd $\leftarrow s$ ▷ Copy the syndome.
2: $\bar{e} \leftarrow \widehat{e}$, $\bar{s} \leftarrow s$ ▷ Initialize the return values.

3: th $\leftarrow$ ComputeThreshold(iter$^{\text{Out}}$, s) ▷ Get the threshold value.
4: **for** $i = 0$ **to** $n_0 - 1$ **do**
5: **for** offset $= 0$ **to** $p - 1$ **do**
6: upc $\leftarrow 0$ ▷ Initialize the unsatisfied parity-check counter.
7: **for** $k = 0$ **to** $v - 1$ **do**
8: **if** GetSyndromeBit(currSynd, (offset + Htr$[i][k]$) (mod p)) $= 1$ **then**
9: upc $\leftarrow$ upc $+ 1$
10: **end if**
11: **if** upc $\geq$ th **then**
12: $\bar{e}_{i \cdot p + \text{offset}} \leftarrow \bar{e}_{i \cdot p + \text{offset}} \oplus 1$ ▷ Update the error vector.
13: **for** $k = 0$ **to** $v - 1$ **do**
14: idx $\leftarrow$ (Htr$[i][k]$ + offset) (mod p)
15: $\bar{s}_{\text{idx}} \leftarrow \bar{s}_{\text{idx}} \oplus 1$ ▷ Update syndrome.
16: **end for**
17: **end if**
18: **end for**
19: **end for**
20: **end for**
21: **return** $\{\bar{e}, \bar{s}\}$

Based on these two algorithms, the decoding algorithm for LEDAcrypt is performed by using a combination of these two iterations to perform a complete pass of a bit-flipping algorithm. Based on some input parameters, namely the number of in-place iterations followed by the number of out of place iterations, the algorithm is as follows.

Algorithm 3.4.4 (LEDAcrypt Decoder [9, p.47])**.**

Input: (s): QC-LDPC syndrome, binary vector of size p.

(Htr): The transposed parity-check matrix, represented as an $n_0 \times v$ integer matrix $H = [H_0|H_1|\dots|H_{n_0-1}]$.

($\widehat{e}$): The initial error vector estimate, binary vector of size $n = n_0 p$.

(imax$^{\text{In}}$): The maximum number of in-place iterations to be executed.

(imax$^{\text{Out}}$): The maximum number of out-of-place iterations to be executed.

Output: (e): updated error vector estimate.

(DecodeOK): error has been decoded properly.

$\widehat{e} \leftarrow 0_n$ ▷ Initialize the error vector.
for $\text{iter}^{\text{In}} = 0$ **to** $\text{imax}^{\text{In}} - 1$ **do**
    if $s = 0$ **then**
        break ▷ Remove this for constant time implementations.
    end if
    $\{\widehat{e}, s\} \leftarrow \text{RandomizedInPlaceIteration}(\widehat{e}, \text{Htr}, s, \text{iter}^{\text{In}})$
end for
for $\text{iter}^{\text{Out}} = 0$ **to** $\text{imax}^{\text{Out}} - 1$ **do**
    if $s = 0$ **then**
        break ▷ Remove this for constant time implementations.
    end if
    $\{\widehat{e}, s\} \leftarrow \text{OutOfPlaceIteration}(\widehat{e}, \text{Htr}, s, \text{iter}^{\text{Out}})$
end for
if $s = 0$ **then**
    DecodeOk = **true**
else
    DecodeOk = **false**
end if
return $\{\bar{e}, \text{DecodeOk}\}$

The in-place iterations allows for better upper-bound on the correction capability of the algorithm whereas the out-of-place iterations allows for more efficiency as they are more amenable to parallel processing. Having a tight bound on the number of iterations that are allowed overall keeps information from leaking through timing differences in different runs of decode.

3.4.2 System Parameters

LEDAcrypt is based on a QC-LDPC code $\mathcal{C}(pn_0, pk_0)$ with generator and parity-check matrices consisting of $p \times p$ circulant sub-matrix blocks. The parity-check matrix H consists of $r_0 \times n_0$ sparse $p \times p$, and the generator matrix G consists of $k_0 \times n_0$. These forms are as in Definitions 2.5.4, 2.5.5 and 2.5.3.

LEDAcrypt is specified for $\lambda = \{1, 3, 5\}$ which is defined as the computation levels required to brute-force decrypt an AES encryption of $\{128, 192, 256\}$ bits, respectively. These are the same defintions as we have seen used in BIKE and Classic McEliece.

The LEDAcrypt Parameter Generation algorithms requires a subroutine NextPrime(x) that returns the first prime p greater than x such that $\text{ord}_2(p) = p-1$, and subroutines C-ISD$_{sdp}(n,r,t)$, C-ISD$_{cfp}(n,r,t)$, Q-ISD$_{sdp}(n,r,t)$ and Q-ISD$_{cfp}(n,r,t)$ that return the costs of the syndrome decoding and codeword finding ISD procedures on a classical and quantum computers respectively. The function Compute_DFR_Ok(n_0,v,t) ensures that a sufficient DFR can be found for the chosen parameters.

Algorithm 3.4.5 (LEDAcrypt-Parameter Setup [5, p.73])**.**

Input: (λ_c, λ_q): The target security levels against classical and quantum attacks.

(ϵ): Either $\log_2(\text{DFR})$ for IND-CCA2, or an enlargement factor for the heristic estimate of a suitable DFR for IND-CPA.

(n_0): The number of circulant blocks of the $p \times n_0 p$ parity-check matrix H of the code.

Output: (p): The size of a circulant block.

(t): The number of errors.

(v): The weight of a column vector of the parity matrix H.

(τ): The number of errors to be corrected with certainty by a single iteration of the decoder.

$p \leftarrow$ starting_prime.
repeat
  $n \leftarrow n_0 p, k \leftarrow (n_0-1)p, r \leftarrow p.$
  $t \leftarrow 0.$
  repeat
    $t \leftarrow t+1$
  until $(t \geq r \vee (\text{C-ISD}_{sdp}(n,r,t) \geq 2^{\lambda_c} \wedge \text{ Q-ISD}_{sdp}(n,r,t) \geq 2^{\lambda_q}))$
  $v \leftarrow 1$
  repeat
    $v \leftarrow v+2$
    $\text{SecureOk} \leftarrow \left(\frac{\text{C-ISD}_{cfp}(n_0 p,p,2v)}{p\binom{n_0}{2}} \geq 2^{\lambda_c}\right) \wedge \left(\frac{\text{Q-ISD}_{cfp}(n_0 p,p,2v)}{p\binom{n_0}{2}} \geq 2^{\lambda_q}\right)$
    $\text{SecureOK} \leftarrow \text{SecureOK} \wedge \left(\frac{\text{C-ISD}_{cfp}(2p,p,2v)}{n_0 p} \geq 2^{\lambda_c}\right)$
  $\wedge \left(\frac{\text{Q-ISD}_{cfp}(2p,p,2v)}{n_0 p} \geq 2^{\lambda_q}\right)$
    $\text{SecureOk} \leftarrow \text{SecureOk} \wedge \left(\frac{\text{C-ISD}_{cfp}(n_0 p,(n_0-1)p,n_0 v)}{p} \geq 2^{\lambda_c}\right)$

$\wedge \left(\frac{\text{Q-ISD}_c fp(n_0 p,(n_0-1)p,n_0 v)}{p} \geq 2^{\lambda_q} \right)$

until (SecureOk = true $\vee$ $vn_0 \geq p$)

if (SecureOk = **true**) **then**

$p, \tau \leftarrow$ Compute_DFR_Ok(n_0, v, t)

end if

until p, v, t do not change

After setting the values of (n, k, r), the algorithm sets t, the error correction parameter to the smallest value that gives sufficient security (2^{λ}) for the best classical and quantum ISD algorithms for solving the Syndrome Decoding problem. Then, when the value of t is set, it starts calculating the size of the circulant matrix v to determine the size of the parity-check matrix H. The algorithm checks the security resistance to the Code Finding problem again, given the best classical and quantum ISD attacks on that problem. The value of v is incremented by 2, due to Theorem 2.5.8 which states that v must be odd for the array to be non-singular.

3.4.3 Key Generation

For LEDACrypt, there are two flavours of encode and decode that need to be considered. The first is an analogue to the McEliece cryptosystem of Example 3.1.7 and the second is an analogue to the Niederreiter cryptosystem of Example 3.1.8. The Key Generation is different for each of these systems, but, the LEDACrypt-KEM only requires the Niederreiter version as the key can be passed through the syndrome and there is no associated message.

The Key Generation algorithm generates two random matrices H, Q to serve as the private key. The matrix $H = [H_0, H_1, \ldots, H_{n_0-1}]$, is structured as a $1 \times n_0$ circulant $p \times p$ blocks with a constant weight $w(H_i) = d_v, 0 \leq i < n_0$. The matrix Q is structured as $n_0 \times n_0$ circulant $p \times p$ blocks. The weights of each block of Q define a circulant matrix of integers denotes as $w(Q)$ such that the sum of all elements in any row or column of Q is the same $m = \sum_{i=0}^{n_0-1} m_i$. The choice of weights $m_i, 0 \leq i < n_0$ is constrained such that Q is invertible as per Theorem 2.5.8, even if a single block Q_{ij} may not be. The private key is then calculated by producing the product of the matrices $L = HQ$ which is a $1 \times n_0$ matrix of $p \times p$ circulant blocks with the same rank as H, $\text{rank}(L) = \text{rank}(H) = p$. Thus, block of L has the same rank (p) and is invertible. The private key M consists of a combination the inverses of the blocks

of L and identity matrices depending on the flavor of the algorithm (McEliece, or Niederreiter).

The seed length l should be large enough to ensure that a sufficient security level is attained.

Algorithm 3.4.6 (LEDAcrypt KeyGen$^{\text{Nie}}$ [9, p.22])**.**

Input: $(p > 2)$: A prime such that $\text{ord}_p(2) = p - 1$, $n_0 \geq 2$.

Output: $sk^{\text{Nie}}, pk^{\text{Nie}}$

1: Generate a seed $\xleftarrow{\$} \mathbb{F}_2^l$ uniformly at random.

2: Generate the $\{H, Q\}$ matrices from this seed.

▷ $H = [H_0, \ldots, H_{n_0-1}]$ where H_i is a $p \times p$ circulant block with $w(H_i) = d_v$, $0 \leq i < n_0$ and Q is an $n_0 \times n_0$ block matrix with $w([Q_{1,0}, \ldots, Q_{1,n_0-1}]) = m$, $0 \leq i \leq n_0$.

3: $L \leftarrow HQ$

▷ $L = [L_0, L_1, \ldots, L_{n_0-1}]$ where each $L_j = \sum_i H_i Q_{ij}$ is a $p \times p$ circulant block.

4: **if** $\exists\ 0 \leq j < n_0$ such that $(L_j) \neq d_v \times m$ **then**

5: **goto** 2

6: **end if**LInv $\leftarrow$ ComputeInverse(L_{n_0-1})

7: **for** $i = 0$ **to** $n_0 - 2$ **do**

8: $M_i \leftarrow \text{LInv}[L_i]$. ▷ M_i is a $p \times p$ circulant block.

9: **end for**

10: $pk^{\text{Nie}} \leftarrow [M_0 | \ldots | M_{n_0-2} | I]$.

11: $sk^{\text{Nie}} \leftarrow \{H, Q\}$.

12: **return** $sk^{\text{Nie}}, pk^{\text{Nie}}$

Algorithm 3.4.7 (LEDAcrypt KeyGen$^{\text{McE}}$ [9, p.22])**.**

Input: $(p > 2)$: A prime such that $\text{ord}_p(2) = p - 1$, $n_0 \geq 2$.

Output: $sk^{\text{Nie}}, pk^{\text{Nie}}$

1: Generate a seed $\xleftarrow{\$} \mathbb{F}_2^l$ uniformly at random.

2: Generate the $\{H, Q\}$ matrices from this seed.

▷ $H = [H_0, \ldots, H_{n_0-1}]$ where H_i is a $p \times p$ circulant block with $w(H_i) = d_v$, $0 \leq i < n_0$ and Q is an $n_0 \times n_0$ block matrix with $w([Q_{1,0}, \ldots, Q_{1,n_0-1}]) = m$, $0 \leq i \leq n_0$.

3: $L \leftarrow HQ$

▷ $L = [L_0, L_1, \ldots, L_{n_0-1}]$ where each $L_j = \sum_i H_i Q_{ij}$ is a $p \times p$ circulant block.

4: **if** $\exists\ 0 \leq j < n_0$ such that $(L_j) \neq d_v \times m$ **then**
5: **goto** 2
6: **end if**
7: LInv $\leftarrow$ ComputeInverse(L_{n_0-1})
8: **for** $i = 0$ **to** $n_0 - 2$ **do**
9: $M_i \leftarrow$ LInv$[L_i]$. ▷ M_i is a $p \times p$ circulant block.
10: **end for**
11: $Z \leftarrow \text{diag}([I, \ldots I])$, ▷ a $p(n_0 - 1) \times p(n_0 - 1)$ identity matrix, composed as a diagonal block matrix with $n_0 - 1$ replicas of I, where I is a $p \times p$ identity matrix.
12: $pk^{\text{McE}} \leftarrow [Z|[M_0|\ldots|M_{n_0-2}]^T]$.
13: $sk^{\text{McE}} \leftarrow \{H, Q\}$.
14: **return** $sk^{\text{McE}}, pk^{\text{McE}}$

Having $p > 2$ such that $\text{ord}_p(2) = p - 1, n_0 \geq 2$ ensures that Q is non-singular by Theorem 2.5.9. McEliece keys are used for the LEDACrypt-PKC. The Niederreiter keys are used for the LEDACrypt-KEM Encapsulate and Decapsulate functions.

3.4.4 Encapsulate

As with the key generation function, there are two flavours of encrypt that we need to consider for encapsulation. The Niederreiter version is used in KEM encapsulation.

Niederreiter encryption takes an error vector with weight t asserted bits as a message and computes the syndrome

$$s = [M_0|M_1|\ldots|M_{n_0-2}|I]e^T$$

as the ciphertext message. This error vector can be used to generate a shared secret key. McEliece encryption takes a message $\mathbf{m}$ and an error vector of weight t and computes the ciphertext

$$\mathbf{c} = [e_0 \mid \ldots \mid e_{n_0-2} \mid e_{n_0-1}] + \left[m_0 \mid \ldots \mid m_{n_0-2} \mid \sum_{j=0}^{n_0-2} m_j M_j \right].$$

hence it is useful for passing encrypted messages along with the key generating error vector [9, p.25].

Algorithm 3.4.8 (LEDAcrypt Encrypt$^{\text{Nie}}$ [8, p.21])**.**

Input: ($\mathbf{e} \in \mathbb{F}_2^{pn_0}$): A plaintext message with $w(\mathbf{e}) = t$ where each $\mathbf{e}_j$ is a $1 \times p$ vector for $0 \leq j < n_0$.

($pk^{\text{Nie}} = [M_0|\ldots|M_{n_0-2}|I]$): The public key .

($p > 2$): A prime such that $\text{ord}_p(2) = p - 1$, $n_0 \geq 2$.

Output: A syndrome $s \in \mathbb{F}_2^p$

1: $s \leftarrow 0_{p\times 1}$
2: **for** $j = 0$ **to** $n_0 - 1$ **do**
3: $\quad s \leftarrow s + M_j\mathbf{e}_j^T$.
4: **end for**
5: $s \leftarrow s + \mathbf{e}_{n_0-1}^T$.
6: **return** s

Algorithm 3.4.9 (LEDAcrypt Encrypt$^{\text{McE}}$ [8, p.21])**.**

Input: ($\mathbf{m} = [\mathbf{m}_0, \ldots, \mathbf{m}_{n_0-2}]$): A plaintext message where each $\mathbf{m}_j$ is a $1 \times p$ vector for $0 \leq j < n_0$.

($\mathbf{e} \in \mathbb{F}_2^{pn_0}$): A plaintext message with $w(\mathbf{e}) = t$ where each $\mathbf{e}_j$ is a $1 \times p$ vector for $0 \leq j < n_0$.

($pk^{\text{Nie}} = [M_0|\ldots|M_{n_0-2}|I]$): The public key .

($p > 2$): A prime such that $\text{ord}_p(2) = p - 1$, $n_0 \geq 2$.

Output: ($\mathbf{c} = [\mathbf{c}_0, \ldots, \mathbf{c}_{n_0-1}]$): A ciphertext in which each $\mathbf{c}_j$ is a $1 \times p$ vector for $0 \leq j < n_0$.

1: blk $\leftarrow 0_{p\times p}$.
2: **for** $j = 0$ **to** $n_0 - 2$ **do**
3: $\quad$ blk $\leftarrow$ blk $+\mathbf{m}_j^T M_j$.
4: **end for**
5: $\mathbf{c} \leftarrow [\mathbf{m}_0, \mathbf{m}_1, \ldots, \mathbf{m}_{n_0-2}, \text{blk}]$
6: **for** $j = 0$ **to** $n_0 - 1$ **do**
7: $\quad \mathbf{c}_j \leftarrow \mathbf{c}_j + \mathbf{e}_j$.
8: **end for**
9: **return c**

This method is based on the Fujisaki-Okamoto conversion of Definition 3.1.18 using $\mathcal{E}^{Nie}$ and $\mathcal{D}^{Nie}$ as the underlying public key encryption and decryption algorithms.

The Encapsulation step requires several different hash functions: the Extensible Output Function (XOF($\mathbf{x}$)), the Key Derivation Function (KDF($\mathbf{x}$)), and a cryptographic hash function HASH($\mathbf{e}$). As in the BIKE parameter generation, these hash functions are of three types.

A number of hash functions are in use here, HASH_{XOF} represents a pre-image resistant extensible output function and HASH_{KDF} represents a key derivation function. The LEDAcrypt specification calls for HASH_{XOF} function to be implemented as SHAKE-128 for $\lambda = 1$ and SHAKE-256 for $\lambda = 3,$ or 5, which extends a seed of size l into a 128 or 256 bit error vector. SHA-3 with an output size of 256, 384 or 512 bits is used as the KDF and HASH for security levels $\lambda = 1, 3,$ or 5 respectively. The KDF and HASH functions differ in that a byte of all zeros is prefixed to the KDF data and a byte of all ones is prefixed to the HASH data before running the SHA-3. The true random number generator (TRNG) in the specification is a CTR-DBRG (a NIST standard deterministic random bit generator which uses AES-256 as a block cipher [10]). The seed length l is $192, 256$ and 320 which is based on security level λ. [8, p.71]

Algorithm 3.4.10 (LEDAcrypt Encapsulate [8, p.21])**.**

Input: (pk): The public key.
($\lambda = 1, 3$ or 5): A security level specifies the length l of the seed.
(XOF, KDF, HASH) HASH functions.

Output: ($\mathbf{k}$): An ephemeral key.
($\mathbf{c}$): An encapsulated secret key.
($\mathbf{x}$): A tag.

1: Seed $\leftarrow$ TRNG(l). ▷ Generate a seed $\xleftarrow{\$}$ $\mathbb{F}_2^l$ uniformly at random.
2: $\mathbf{e} \leftarrow \text{HASH}_{\text{XOF}}(\text{Seed})$. ▷ Generate $\mathbf{e}$ by hashing the seed using the extensible output function (XOF).
3: $\mathbf{k} = \text{HASH}_{\text{KDF}}(\mathbf{e})$ ▷ Derive a key using a pre-image resistant key derivation function (KDF) on $\mathbf{e}$, (i.e. another hash function) on $\mathbf{e}$.
4: $\mathbf{c} \leftarrow \mathcal{E}_{pk}(\mathbf{e})$ ▷ Use the encrypt function to encrypt $\mathbf{e}$ using the public key.
5: mask$\leftarrow$ HASH($\mathbf{e}$) ▷ Hash $\mathbf{e}$ to produce a mask.
6: $\mathbf{x} \leftarrow$ seed $\oplus$ mask. ▷ Set the tag x to use in the verification step to ensure that the key passed successfully.
7: **return** $\mathbf{k}, \mathbf{c}, \mathbf{x}$

3.4.5 Decapsulate

As in the case on Encapsulate, we have two versions of the decoding to choose from, one based on McEliece and another based Niederreiter. The decoding system must match that which was used to encode the message and for the KEM, this should be the Niederreiter version. The error vector $\mathbf{e}$ is to be determined and used to decapsulate the shared secret key K.

Algorithm 3.4.11 (LEDAcrypt Decrypt$^{\text{Nie}}$ [8, p.25])**.**

Input: $(p > 2)$: A prime such that $\text{ord}_p(2) = p - 1$, $n_0 \geq 2$.
$(s \in \mathbb{F}_2^p)$: A syndrome.
$(sk^{\text{Nie}} = \{H, Q\})$: The secret key.

Output: $(\mathbf{e})$: The error as a sequence of n_0 binary vectors $\mathbf{e}_j \in \mathbb{F}_2^p$

1: $L \leftarrow HQ$, such that $L = [L_0|\ldots|L_{n_0-1}]$.
2: $\mathbf{s}' \leftarrow L_{n_0-1}s$
3: $\mathbf{e} = \text{Decode}(\mathbf{s}', sk^{\text{Nie}})$. ▷ Decode using Bitflip Algorithm 3.4.1.
4: **if** $\mathbf{e} = \perp$ **then**
5: Decode failed.
6: **end if**
7: **return** $\mathbf{e}$ (Either $\perp$ or the decoded error matrix).

Algorithm 3.4.12 (LEDAcrypt Decrypt$^{\text{McE}}$ [8, p.25])**.**

Input: $(p > 2)$: A prime such that $\text{ord}_p(2) = p - 1$, $n_0 \geq 2$.
$(\mathbf{c} = [c_0, \ldots, c_{n_0-1}] \in \mathbb{F}_2^{pn_0})$: An error affected codeword where each $\mathbf{c}_j$ is a $1 \times p$ vector with $0 \leq j < n_0$.
$(sk^{\text{McE}} = \{H, Q\})$: The secret key.

Output: $(\mathbf{m} = [m_0, \ldots, m_{n_0-2}])$: The plaintext, a sequence of $n_0 - 1$ binary vectors of size $1 \times p$.
$(\mathbf{e})$: The error as a sequence of n_0 binary vectors $\mathbf{e}_j \in \mathbb{F}_2^p$ or $\perp$.

1: $L \leftarrow HQ$, such that $L = [L_0|\ldots|L_{n_0-1}]$.
2: $\mathbf{s} \leftarrow L\mathbf{c}^T$
3: $\mathbf{e} = \text{Decode}(\mathbf{s}, sk^{\text{McE}})$. ▷ Decode using Bitflip Algorithm 3.4.1.
4: **if** $\mathbf{e} \neq \perp$ **then**
5: **for** $j = 0$ **to** $n_0 - 1$ **do**
6: $\mathbf{m}_j \leftarrow \mathbf{c}_j + \mathbf{e}_j$

7: **end for**
8: **else**
9: $\mathbf{e} \leftarrow \perp, \mathbf{m} \leftarrow \perp$.
10: **end if**
11: **return** $\mathbf{e}, \mathbf{m}$.

Again, in using LEDACrypt-KEM, the choice of decoding algorithms is $\mathcal{D}^{Nie}$, the decapsulation algorithm is based on determining the error using the passed syndrome. The hash functions used must also be identical to those used in encapsulation.

Algorithm 3.4.13 (LEDAcrypt Decapsulate [8, p.27])**.**

Input: $(sk = \{H, Q\})$: The secret key.
(LongTermSeed): A secret random longterm seed.
$(\mathbf{c})$: The encapsulated secret.
$(\mathbf{x})$: The tag.
Output: $(\mathbf{k})$: The shared secret key.

1: tmp $\leftarrow$ Concatenate(LongTermSeed, $\mathbf{c}$)
2: mask $\leftarrow$ HASH($\mathbf{e}$)
3: seed $\leftarrow$ mask$\oplus\mathbf{x}$
4: $\mathbf{e}' \leftarrow \text{HASH}_{\text{XOF}}(\text{seed})$.
5: **if** $\mathbf{e} \neq \perp$ and $w(\mathbf{e}) = t$ and $\mathbf{e}' = \mathbf{e}$ **then**
6: $\mathbf{k} \leftarrow \text{HASH}_{\text{KDF}}(\mathbf{e})$.
7: **else**
8: $\mathbf{k} \leftarrow \text{HASH}_{\text{KDF}}(\text{tmp})$
9: **end if**
10: **return k**

3.4.6 Attacks on System

As in the other KEMs we have looked at so far, the security of LEDAcrypt and the choice of parameters are based on the best variations of Gallager's ISD of Algorithm 3.2.11. In the parameter generation Algorithm 3.4.5, the trial values for the p, t and v parameters are passed to algorithms which determine the computational difficulty of solving one of the security base problems using best variation of ISD in both classical and quantum variations.

There are 8 ISD solving algorithms that are considered in choosing parameters for LEDACrypt systems. These ISD solvers are customize to solve the following problems

- C-SDP, Q-SDP are designed to solve the syndrome decoding problem for key recovery;
- C-CFP-1, Q-CFP-1 are designed to solve the codeword finding problem in the hidden LDPC/MDCP code;
- C-CFP-2, Q-CFP-2 are designed to solve the codeword finding problem in the length-reduced hidden LDPC/MDPC code;
- C-CFP-3, Q-CFP-3 are designed to solve the codeword finding problem on the dual code of the hidden code.

The CFP-3 is the most effective for parameter sets with larger $n_0 = 3, 4$ whereas the key recovery attacks are almost identical for all parameter sets with $n_0 = 2$.

For an exhaustive search on LEDAcrypt keys based on Lee and Brickell's ISD, [39] the correct key is found in

$$\left(\frac{\binom{n}{w}}{\binom{k}{x}\binom{r}{w-x}} \right) (c_{test})$$

attempts where c_{test} is the cost for testing a potential key and x is a free variable to be optimized [9, p.43].

In April 2020, a weak-key attack was published on LEDAcrypt by Apon et al. [2] showed that for an earlier revision of LEDAcrypt the product structure $L = HQ$ for the public key allowed H and Q to be guessed separately, which reduced the security level of the chosen parameters to well below the required λ. Specifically, 1 in $2^{47.72}$ of LEDAcrypt's keys could be recovered using only $2^{18.72}$ guesses giving $47.72 + 18.72 = 66.44$ bits of security at the 256-bit level ($\lambda = 5$) [9, p.43].

The attack exploits the product structure of $L = HQ$ to first define a class of extremely weak keys. There is a large enough number of such keys to be a concern. An example of such weak keys are keys where the polynomials L_0 and L_1 are of degree at most $p/2$. Such keys can be found in a single iteration of a modified version of the ISD algorithm (Algorithm 3.2.11)

For our modified ISD, the attacker Eve selects the information set S so that it consists of the last $\frac{p-1}{2}$ columns of the first block of the public key M and the last $\frac{p+1}{2}$ columns of the second block. In the case of one of these extremely weak keys,

Eve guesses the top row of L to all zeros in the information set and solves a (nonzero) linear combination of rows of M thereby extracting the key.

A sufficient condition for a key to be extremely weak in this manner is for the degree each of the polynomials $H_0, H_1, Q_{0,0}, Q_{0,1}, Q_{1,0}, Q_{1,1}$ to be no more than $p/4$. Each of the $2m_0+2m_1+2d_v$ nonzero coefficients of these polynomials has a probability $1/4$ of having degree less than $p/4$, therefore weak keys of this form occur with probability $(1/4)^{2m_0+2m_1+2d_v}$ [2, p.16, 17].

These attacks were countered in the newer revision by increasing n_0 or ensuring that the weight of H and Q are unbalanced. [9, p.43].

3.4.7 Security Analysis

Some of the recommended and implemented sets of parameters given in the following tables are as follow:

- $Cat.$ the security category target of the parameter list ($\lambda - \{1, 3, 5\}$)
- n_0, the number of circulant blocks of H,
- p, the size of the circulant blocks,
- v, the weight of a column vector of H,
- t, the number of errors that can be corrected by the code,
- τ, the number of errors that can be corrected in a single iteration of the bit-flipper,
- $key\%$ the percentage of keys that are passed successfully during key generation, and
- $\log_2(inst)$ an estimate of the computation complexity (i.e. the log of the cost in instructions or cycles) of the best algorithm to solve the SDP and CFP problems on a classical or quantum computer.

Three different versions of the LEDAdecoder of Algorithm 3.4.4 are considered each of which has $\text{imax}^{\text{in}} + \text{imax}^{\text{out}} = 2$. That is parameter sizes have been calculated for each of the following versions of the decoder.

$imax^{in}$	$imax^{out}$
2	0
1	1
0	2

These values are shown below as i^{in} and i^{out}, respectively.

For $imax^{in} = 0$ and $imax^{out} = 2$, the security of the following parameters have been calculated.

Cat.	DFR	i^{in}	i^{out}	n_0	p	v	t	τ	$Key\%Ok$	$\log_2(inst)$
$\lambda-1$	2^{-64}	0	2	2	23,371	71	130	10	85.3	93.4
$\lambda-1$	2^{-64}	0	2	3	16,067	79	83	9	86.5	93.9
$\lambda-1$	2^{-64}	0	2	4	13,397	83	66	8	93.7	94.4
$\lambda-1$	2^{-128}	0	2	2	28,277	69	129	11	68.2	92.0
$\lambda-1$	2^{-128}	0	2	3	19,709	79	82	10	66.5	94.5
$\lambda-1$	2^{-128}	0	2	4	16,229	83	65	9	63.5	95.0
$\lambda-3$	2^{-64}	0	2	2	40,787	103	195	13	81.9	126.5
$\lambda-3$	2^{-64}	0	2	3	28,411	117	124	11	99.0	128.4
$\lambda-3$	2^{-64}	0	2	4	22,901	123	98	11	61.4	128.7
$\lambda-3$	2^{-192}	0	2	2	52,667	103	195	15	59.8	127.2
$\lambda-3$	2^{-192}	0	2	3	36,629	115	123	13	81.9	127.3
$\lambda-3$	2^{-192}	0	2	4	30,803	123	98	12	75.9	129.5
$\lambda-5$	2^{-64}	0	2	2	61,717	137	261	17	55.2	161.3
$\lambda-5$	2^{-64}	0	2	3	42,677	153	165	14	96.4	160.9
$\lambda-5$	2^{-64}	0	2	4	35,507	163	131	13	95.9	162.9
$\lambda-5$	2^{-256}	0	2	2	83,579	135	260	18	51.6	160.2
$\lambda-5$	2^{-256}	0	2	3	58,171	153	165	16	65.2	161.6
$\lambda-5$	2^{-256}	0	2	4	48,371	161	131	15	78.9	163.6

For $\text{imax}^{\text{in}} = 1$ and $\text{imax}^{\text{out}} = 1$, the security of the following parameters have been calculated.

Cat.	DFR	i^{in}	i^{out}	n_0	p	v	t	τ	$Key\%Ok$	$\log_2(inst)$
$\lambda - 1$	2^{-64}	1	1	2	21,701	71	130	10	75.56	93.2
$\lambda - 1$	2^{-64}	1	1	3	15,053	79	83	9	75.74	93.7
$\lambda - 1$	2^{-64}	1	1	4	12,739	83	66	8	89.69	94.3
$\lambda - 1$	2^{-128}	1	1	2	27,077	69	130	11	62.61	93.8
$\lambda - 1$	2^{-128}	1	1	3	19,541	79	82	10	45.10	94.4
$\lambda - 1$	2^{-128}	1	1	4	15,773	83	65	9	53.33	94.9
$\lambda - 3$	2^{-64}	1	1	2	37,813	103	196	13	70.6	126.3
$\lambda - 3$	2^{-64}	1	1	3	26,597	117	124	11	97.4	128.2
$\lambda - 3$	2^{-64}	1	1	4	22,229	123	98	11	57.0	128.7
$\lambda - 3$	2^{-192}	1	1	2	50,363	103	195	15	50.1	127.1
$\lambda - 3$	2^{-192}	1	1	3	35,419	115	123	13	73.8	129.0
$\lambda - 3$	2^{-192}	1	1	4	29,221	123	98	13	61.2	129.4
$\lambda - 5$	2^{-64}	1	1	2	58,171	137	262	17	53.1	161.2
$\lambda - 5$	2^{-64}	1	1	3	40,387	153	165	14	90.8	162.4
$\lambda - 5$	2^{-64}	1	1	4	33,493	163	131	13	88.8	162.7
$\lambda - 5$	2^{-256}	1	1	2	81,773	135	260	17	87.0	160.1
$\lambda - 5$	2^{-256}	1	1	3	56,731	153	165	16	52.9	161.6
$\lambda - 5$	2^{-256}	1	1	4	47,459	161	131	15	73.0	163.6

For $\text{imax}^{\text{in}} = 2$ and $\text{imax}^{\text{out}} = 0$, the security of the following parameters have been calculated.

Cat.	DFR	i^{in}	i^{out}	n_0	p	v	t	τ	$Key\%Ok$	$\log_2(inst)$
$\lambda - 1$	2^{-64}	2	0	2	23,371	71	130	10	85.3	93.4
$\lambda - 1$	2^{-64}	2	0	3	16,067	79	83	9	86.5	93.9
$\lambda - 1$	2^{-64}	2	0	4	13,397	83	66	8	93.7	94.4
$\lambda - 1$	2^{-128}	2	0	2	28,277	69	129	11	68.2	92.0
$\lambda - 1$	2^{-128}	2	0	3	19,709	79	82	10	66.6	94.5
$\lambda - 1$	2^{-128}	2	0	4	16,229	83	65	9	63.5	95.0
$\lambda - 3$	2^{-64}	2	0	2	40,787	103	195	13	81.9	126.5
$\lambda - 3$	2^{-64}	2	0	3	28,411	117	124	11	99.0	128.4
$\lambda - 3$	2^{-64}	2	0	4	22,901	123	98	11	61.4	128.7
$\lambda - 3$	2^{-192}	2	0	2	52,667	103	195	15	59.8	127.2
$\lambda - 3$	2^{-192}	2	0	3	36,629	115	123	13	91.9	127.3
$\lambda - 3$	2^{-192}	2	0	4	30,803	123	98	12	75.9	129.5
$\lambda - 5$	2^{-64}	2	0	2	61,717	137	261	17	55.2	161.3
$\lambda - 5$	2^{-64}	2	0	3	42,677	153	165	14	96.4	160.9
$\lambda - 5$	2^{-64}	2	0	4	35,507	163	131	13	95.9	162.9
$\lambda - 5$	2^{-256}	2	0	2	83,579	135	260	18	51.6	160.2
$\lambda - 5$	2^{-256}	2	0	3	58,171	153	165	16	65.2	161.6
$\lambda - 5$	2^{-256}	2	0	4	48,371	161	131	15	78.9	163.6

Each of these security levels are equal or greater than those required by the security category in column 1. In most cases (except for $n_0 = 2$) the most efficient attack is the Q-CFP-3 which is a quantum attack on codeword finding on the dual code of the hidden code. In the case of $n_0 = 2$, the most efficient attack is Q-CFP-2 which

is a quantum attack on the codeword finding problem in the length-reduced hidden LDPC code.

3.5 Hamming Quasi-Cyclic

The Hamming Quasi-Cyclic cryptosystem is another code-based IND-CPA system based on the hardness of the problem of syndrome decoding. The Fujisaki-Okomoto transformation does allow conversion to an IND-CCA2 hybrid type cryptosystem. There are two flavours of the KEM both based on building a secure KEM using a combination of two less secure codes. In the case of HQC, the codes are repetition codes and BCH codes which are combined using the tensor product and in the case of HQC-RMRS the codes are Reed-Muller and Reed-Solomon codes which are combined using a concatenation operation. HQC has small public key sizes and efficient decoding algorithms based on well known codes.

The following notations are used during the descriptions of HQC. Let

$$S_w^n(\mathbb{F}_2) = \{\mathbf{v} \in \mathbb{F}_2^n | w(\mathbf{v}) = w\}$$

be the set of all binary vectors of length n and weight w. Let $\mathcal{R} = \mathbb{F}_2[x]/(x^n - 1)$ be a polynomial ring. Let $\mathcal{V}$ be a vector space of dimension n over $\mathbb{F}_2$ then $\mathcal{V}$ can be represented as polynomials in $\mathcal{R}$ or row vectors of a matrix by Theorem 2.5.6. A prime integer n is primitive if the polynomial $x^n - 1/(x-1)$ is irreducible in $\mathcal{R}$. For $\mathbf{u}, \mathbf{v} \in \mathcal{V}$ the product $\mathbf{u} \cdot \mathbf{v}$ is defined as

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{w} \in \mathcal{V},$$

where for $0 \leq k < n$

$$w_k = \sum_{i+j \equiv k \pmod n} u_i v_j,$$

which is the standard polynomial multiplication in $\mathcal{R}$.

Definition 3.5.1 (Circulant Matrix $\mathbf{rot}(\mathbf{v})$ [48, p.8])**.** Let $\mathbf{v} = (v_0, \ldots, v_{n-1}) \in \mathbb{F}_2^n$.

The *circulant matrix* induced by $\mathbf{v}$ is defined and denoted as follows:

$$\mathbf{rot}(\mathbf{v}) = \begin{pmatrix} v_0 & v_{n-1} & \cdots & v_1 \\ v_1 & v_0 & \cdots & v_2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{n-1} & v_{n-2} & \cdots & v_0 \end{pmatrix}$$

The product of any two elements $\mathbf{u}, \mathbf{v} \in \mathcal{R}$ can be expressed as a usual vector-matrix (or matrix-vector) product using the $\mathbf{rot}(\cdot)$ operator as

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u} \times \mathbf{rot}(\mathbf{v})^\top = (\mathbf{rot}(\mathbf{u}) \times \mathbf{v}^\top)^\top = \mathbf{v} \times \mathbf{rot}(\mathbf{u})^\top = \mathbf{v} \cdot \mathbf{u}. \tag{3.2}$$

For example, using a 3×3 matrix with $\mathbf{u} = \begin{pmatrix} u_0 & u_1 & u_2 \end{pmatrix}$, $\mathbf{v} = \begin{pmatrix} v_0 & v_1 & v_2 \end{pmatrix}$, we have

$$\mathbf{rot}(\mathbf{u}) = \begin{pmatrix} u_0 & u_2 & u_1 \\ u_1 & u_0 & u_2 \\ u_2 & u_1 & u_0 \end{pmatrix}, \ \mathbf{rot}(\mathbf{v}) = \begin{pmatrix} v_0 & v_2 & v_1 \\ v_1 & v_0 & v_2 \\ v_2 & v_1 & v_0 \end{pmatrix}$$

Then

$$\begin{aligned} \mathbf{u} \times \mathbf{rot}(\mathbf{v})^\top &= \begin{pmatrix} u_0 & u_1 & u_2 \end{pmatrix} \begin{pmatrix} v_0 & v_1 & v_2 \\ v_2 & v_0 & v_1 \\ v_1 & v_2 & v_0 \end{pmatrix} \\ &= u_0 \begin{pmatrix} v_0 \\ v_2 \\ v_1 \end{pmatrix} + u_1 \begin{pmatrix} v_1 \\ v_0 \\ v_2 \end{pmatrix} + u_2 \begin{pmatrix} v_2 \\ v_1 \\ v_0 \end{pmatrix} \\ &= \begin{pmatrix} u_0 v_0 + u_1 v_1 + u_2 v_2 \\ u_0 v_2 + u_1 v_0 + u_2 v_1 \\ u_0 v_1 + u_1 v_2 + u_2 v_0 \end{pmatrix} \end{aligned}$$

$$
\begin{aligned}
&= \begin{pmatrix} u_0v_0 + u_1v_1 + u_2v_2 \\ u_1v_0 + u_2v_1 + u_0v_2 \\ u_2v_0 + u_0v_1 + u_1v_2 \end{pmatrix} \\
&= v_0 \begin{pmatrix} u_0 \\ u_1 \\ u_2 \end{pmatrix} + v_1 \begin{pmatrix} u_1 \\ u_2 \\ u_0 \end{pmatrix} + v_2 \begin{pmatrix} u_2 \\ u_0 \\ u_1 \end{pmatrix} \\
&= \begin{pmatrix} v_0 & v_1 & v_2 \end{pmatrix} \begin{pmatrix} u_0 & u_1 & u_2 \\ u_1 & u_2 & u_0 \\ u_2 & u_0 & u_1 \end{pmatrix} \\
&= \mathbf{v} \times \mathbf{rot}(\mathbf{u})^\top
\end{aligned}
$$

That is, multiplication of these vectors commutes.

The definition of a circulant matrix here is the transpose of the standard form of Definition 2.5.3, and it is given the notation $\mathbf{rot}(\mathbf{v})$. The standard circulant matrix form is then the transpose again $\mathbf{rot}(\mathbf{v})^\top$ and the multiplication chain in (3.2) alternates between vector and polynomial representations to show that the multiplication commutes.

Definition 3.5.2 (Systematic Quasi-Cyclic Code [48, p.9])**.** A *systematic quasi-cyclic* $[sn, n]$ code of index s and rate $1/s$ is a quasi-cyclic code with an $(s-1)n \times sn$ parity-check matrix of the form:

$$
\mathbf{H} = \begin{bmatrix} \mathbf{I_n} & 0 & \cdots & 0 & \mathbf{A_0} \\ 0 & \mathbf{I_n} & \cdots & 0 & \mathbf{A_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & \mathbf{I_n} & \mathbf{A_{s-2}} \end{bmatrix},
$$

where $\mathbf{A_0}, \dots, \mathbf{A_{s-2}}$ are circulant $n \times n$ matrices.

This systematic form can be generalized to allow for rates of different fractions but HQC codes use rates of 1/2 and 1/3 only. For HQC references to systematic

quasi-cyclic codes refer to codes of rates 1/s only.

There are two forms of HQC under consideration, HQC and HQC-RMRS. The HQC form consists of a code created by combining a repetition code and a BCH code using the Tensor product. The HQC-RMRS form consists of concatenation of a Reed-Muller and a Reed-Solomon code.

Definition 3.5.3 (Tensor Product Code [48, p.21])**.** Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be $[n_1, k_1, d_1]$ and $[n_2, k_2, d_2]$ linear codes over $\mathbb{F}_2$s. The *Tensor Product Code* of $\mathcal{C}_1$ and $\mathcal{C}_2$ denoted as $\mathcal{C}_1 \otimes \mathcal{C}_2$ is defined as the set of all $n_2 \times n_1$ matrices whose rows are codewords of $\mathcal{C}_1$ and whose columns are codewords of $\mathcal{C}_2$. If $\mathbf{G}_1$ and $\mathbf{G}_2$ are the generators of $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively, then

$$\mathcal{C}_1 \otimes \mathcal{C}_2 = \{\mathbf{G}_1\mathbf{X}\mathbf{G}_2 \text{ for } \mathbf{X} \in \mathbb{F}_2^{k_2 \times k_1}\}.$$

The tensor product of two linear codes is a $[n_1n_2, k_1k_2, d_1d_2]$ linear code.

Definition 3.5.4 (BCH Codes used in HQC [48, p.22])**.** For any positive integers $m \geq 3$ and $t \leq 2^{m-1}$, there exists a binary BCH code with the following parameters:

- block length $n = 2^{m-1}$,
- number of parity-check digits $n - k \leq m\delta$, with δ, the correcting capacity of the code and k the number of information bits, and
- minimum distance $d_{min} \leq 2\delta + 1$.

We denote this code by BCH$[n, k, \delta]$. Let α be a primitive element in $\mathbb{F}_{2^m}$, the generator polynomial g of the BCH$[n, k, \delta]$ code is given by:

$$g(x) = \text{lcm}\{\phi_1(x), \phi_2(x), \ldots, \phi_{2\delta}(x)\}$$

with ϕ_i being the minimal polynomial of α_i (see Definition 2.1.15).

The designed distance of the BCH code is $d \leq 2\delta + 1$, and Definition 2.2.3 states that

$$g(x) = \text{lcm}(M^{(b)}(x), M^{(b+1)}(x), \ldots, M^{(b+d-2)}(x)).$$

With $b = 1, d = 2\delta - 1$ this is

$$g(x) = \text{lcm}(M^{(1)}(x), M^{(1+1)}(x), \ldots, M^{(1+2\delta+1-2)}(x)),$$

which implies

$$g(x) = \text{lcm}(M^{(1)}(x), M^{(1+1)}(x), \ldots, M^{(2\delta)}(x)).$$

This converts to the form given above.

In order to fix the dimension $k = 256$, we shorten the generated BCH codes. Shortened BCH codes are then constructed by subtracting constant values from n and k. For example the BCH$[1023, 513, 57]$, a shortened code BCH-S1 is constructed by subtracting 257 from n, and k giving BCH-S1$[766, 256, 57]$.

The following shortened codes are defined for use in HQC.

- BCH-S1$[716, 256, 57]$, and
- BCH-S2$[796, 256, 60]$.

Code shortening does not affect the correcting capacity δ [48, p.23].

Encoding and decoding for BCH codes is covered in Definition 2.2.3 and Algorithm 2.2.5.

Definition 3.5.5 (Repetition Codes used in HQC [41, p.474])**.** A *repetition code* $\mathbb{1}_n$ *word* consists of only one message symbol and $n-1$ control symbols all consisting of the symbol.

Definition 3.5.6 (Repetition decoding in HQC [48, p.2])**.** The decoding algorithm used for the repetition codes in HQC is the *majority decoding*.

$$\text{Decode}(\mathbf{x}) = \begin{cases} 1, & \text{if } \sum_{i=0}^{n-1} x_i \geq \lceil \frac{n+1}{2} \rceil, \\ 0, & \text{otherwise.} \end{cases}$$

The other type of code suggested for use in HQC is the concatenated Reed-Muller/Reed-Solomon Codes.

Definition 3.5.7 (Concatenated Codes [48, p.21])**.** A *concatenated code* consists of an external code $[n_e, k_e, d_e]$ over $\mathbb{F}_q$ and an internal code $[n_i, k_i, d_i]$ over $\mathbb{F}_2$, with $q = 2^k$. We use a bijection between elements of $\mathbb{F}_q$ and the words of the internal code, this way we obtain a transformation

$$\mathbb{F}_q^{n_e} \to \mathbb{F}_2^N$$

where $N = n_e n_i$. The external code is thus transformed into a binary code of parameters $[N = n_e n_i, K = k_e k_i, D \geq d_e d_i]$.

Definition 3.5.8 (Reed-Muller Codes used in HQC [48, p.32])**.** For any positive integers m and r with $0 \leq r \leq m$, there exists a binary r^{th} order Reed-Muller code denoted by $\mathrm{RM}(r, m)$ with the following parameters:

- Code length $= n = 2^m$,
- Dimension $= k = \sum_{i=0}^{r} \binom{m}{i}$, and
- Minimum distance $d_{min} = 2^{m-r}$.

Definition 3.5.9 (Reed-Solomon Codes used in HQC [48, p.28])**.** Let p be a prime number and q be any power of p. A Reed-Solomon code with symbols in $\mathbb{F}_q^n$ has the following parameters:

- Block length $n = q - 1$,
- Number of parity-check digits $n - k = 2\delta$, with δ the correcting capacity of the code and k the number of information bits, and
- Minimum distance $d_{min} = 2\delta + 1$.

We denote this code by $\mathrm{RS}[n, k, \delta]$. Let α be a primitive elements in $\mathbb{F}_q^m$, the generator polynomial $g(x)$ of the $\mathrm{RS}[n, k, \delta]$ code is given by

$$g(x) = (x + \alpha)(x + \alpha^2) \cdots (x + \alpha^{2\delta}).$$

Shortened RS codes are then constructed by subtracting constant values from parameters n and k.

Code	n	k	S-code	diff	n'	k'	δ
RS-1	255	207	RS-S1	175	80	32	24
RS-2	255	211	RS-S2	179	76	32	22
RS-3	255	209	RS-S3	177	78	32	23

In this table, the original RS-$i[n, k, \delta]$, codes are given with their associated RS-S$i[n', k', \delta]$ codes, $1 \leq i \leq 3$. The values for n' and k' are generated by subtracting *diff* from n and k, while the value δ remains unchanged by this shortening.

3.5.1 Security Base Problem

The security of HQC is based on the hard problems that are variations of the problems we have seen before, namely the syndrome decoding (SD) problem of Definition 3.3.1. All of the hard problems described below are variants of this syndrome decoding problem for quasi-cyclic codes of index $s = 2, 3$ with a rate of $1/s = 1/2, 1/3$.

Definition 3.5.10 (s-QCSD Distribution [48, p.10])**.** For positive integers n, w, and s, the *s-QCSD(n, w) Distribution* chooses uniformly at random a parity-check matrix $\mathbf{H} \xleftarrow{\$} \mathbb{F}_2^{(sn-n)\times sn}$ of a systematic QC code $\mathcal{C}$ of index s and rate $1/s$ (see Definition 3.5.2) together with a vector $\mathbf{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_{s-1}) \xleftarrow{\$} \mathbb{F}_2^{sn}$ such that $\omega(\mathbf{x}_i) = \omega$, $0 \leq i < s$ and outputs $(\mathbf{H}, \mathbf{H}\mathbf{x}^\top)$. Recall from Definition 2.1.7 that w represents a norm on $\mathcal{R}$.

Having set up the distribution, and given $\mathbf{H}, \mathbf{H}\mathbf{x}^\top$, the problem is to find the $\mathbf{x}$ which generated it.

Definition 3.5.11 (Computational s-QCSD Problem [48, p.10])**.** For positive integers n, w, s a random parity-check matrix $\mathbf{H}$ of a systematic QC code $\mathcal{C}$ of index s and $\mathbf{y} \xleftarrow{\$} \mathbb{F}_2^{sn-n}$, the *Computational s-Quasi-Cyclic SD Problem (s-QCSD)(n, w)* asks to find $\mathbf{x} = \{x_0, \ldots, x_{s-1}\} \in \mathbb{F}_2^{sn}$ such that $\omega(x_i) = \omega, 0 \leq i < s$, and $\mathbf{y} = \mathbf{x}\mathbf{H}^\top$.

Specifically, HQC uses double and triple circulant codes with systematic generator matrices as described in Definition 3.5.2. The specific problems are outlined below. The matrix $\mathbf{H}$ above is specified as being in systematic form, this is done to make the security analysis easier. There is some probability that any uniformly generated quasi-cyclic code can be reduced to the systematic form. The security analysis is done with this systematic form as it is the less computationally intensive form of the attack.

There are decisional variants of the above problems which attempt to describe the system's indistinguishability (Definition 3.1.12) based attacks.

The systematic form of $\mathbf{H}$ being used for the decisional variants is not that of Definition 3.5.2. Additional constraints (below) are introduced to avoid the attack of Liu, Pan and Xie [42, p.316] whereby a public key can be distinguished from uniform distribution by evaluating h and s at 1.

Definition 3.5.12 (Additional HQC constraints [48, p.11])**.** For $b \in \{0, 1\}$ define a finite set $\mathbb{F}_{2,b}^n = \{\mathbf{h} \in \mathbb{F}_2^{2n\times 3n}$ s.t. $\mathbf{h}(1) = b \pmod 2\}$, i.e. binary vectors of length n and parity b and matrices,

$$\mathbb{F}_{2,b}^{n\times 2n} = \{\mathbf{H} = \begin{pmatrix} \mathbf{I}_n & \mathbf{rot}(\mathbf{h}) \end{pmatrix} \in \mathbb{F}_2^{n\times 2n} \text{ s.t. } \mathbf{h} \in \mathbb{F}_{2,b}^n\}, \text{ and}$$

$$\mathbb{F}_{2,b1,b2}^{2x\times 3x} = \left\{ \mathbf{H} = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{h_1}) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{h_2}) \end{pmatrix} \in \mathbb{F}_2^{n\times 2n} \text{ s.t. } \mathbf{h}_1 \in \mathbb{F}_{2,b_1}^n, \mathbf{h}_2 \in \mathbb{F}_{2,b_2}^n \right\}.$$

Definition 3.5.13 (2-QCSD Distribution (with Parity) [48, p.11])**.** For positive integers n, w and b the *2-QCSD(n, w, b) Distribution* with parity chooses uniformly at random a parity-check matrix $\mathbf{H} \in \mathbb{F}_{2,b}^{n\times 2n}$ together with a vector $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \xleftarrow{\$} \mathbb{F}_2^{2n}$ such that $\omega(\mathbf{x}_1) = \omega(\mathbf{x}_2) = \omega$ and outputs $(\mathbf{H}, \mathbf{H}\mathbf{x}^\top)$.

Definition 3.5.14 (Decisional 2-QCSD Problem (with parity) [48, p.11])**.** Let $\mathbf{h} \in \mathbb{F}_{2,b}^n$, $\mathbf{H} = (\mathbf{I}_n \ \mathbf{rot}(\mathbf{h}))$, and $b' \equiv w + b \times w \pmod 2$. For $\mathbf{y} \in \mathbb{F}_{2,b'}^n$, the *Decisional 2-Quasi-Cyclic SD Problem* with parity *2-DQCSD*(n, w, b) asks to decide with non-negligible advantage whether $(\mathbf{H}, \mathbf{y})$ came from the $2 - QCSD(n, w, b)$ distribution with parity or the uniform distribution over $\mathbb{F}_{2,b}^{n\times 2n} \times \mathbb{F}_{2,b'}^n$.

Definition 3.5.15 (3-QCSD Distribution (with Parity) [48, p.11])**.** For positive integers n, w, b_1 and b_2 the *3-QCSD(n, w, b_1, b_2) Distribution* with parity chooses uniformly at random a parity-check matrix $\mathbf{H} \in \mathbb{F}_{2,b_1,b_2}^{2n\times 3n}$ together with a vector $\mathbf{x} = (x_1, x_2, x_3) \xleftarrow{\$} \mathbf{F}_2^{3n}$ such that $\omega(x_1) = \omega(x_2) = \omega(x_3) = \omega$ and outputs $(\mathbf{H}, \mathbf{H}\mathbf{x}^\top)$.

Definition 3.5.16 (Decisional 3-QCSD Problem (with parity) [48, p.12])**.** Let $\mathbf{h_1} \in \mathbb{F}_{2,b_1}^n, \mathbf{h_2} \in \mathbb{F}_{2,b_2}^n$, $\mathbf{H} = \begin{pmatrix} \mathbf{I}_n & \mathbf{0} & \mathbf{rot}(\mathbf{h_1}) \\ \mathbf{0} & \mathbf{I}_n & \mathbf{rot}(\mathbf{h_2}) \end{pmatrix}$, and $b'_1 \equiv w + b_1 \times w \pmod 2$ and $b'_2 \equiv w + b_2 \times w \pmod 2$. For $\mathbf{y} \in \mathbb{F}_{2,b'}^n$, the *Decisional 3-Quasi-Cyclic SD Problem* with parity *3-DQCSD*(n, w, b_1, b_2) asks to decide with non-negligible advantage whether $(\mathbf{H}, (\mathbf{y_1}, \mathbf{y_2}))$ came from the $3 - QCSD(n, w, b_1, b_2)$ distribution with parity or the uniform distribution over $\mathbb{F}_{2,b_1,b_2}^{2n\times 3n} \times \left(\mathbb{F}_{2,b'_1}^n \times \mathbb{F}_{2,b'_2}^n\right)$.

The IND-CPA security of the HQC system is based on the assumed hardness of the 2- and 3-DQCSD problems of Definitions 3.5.14, and 3.5.16.

The codes defined for HQC are concatenated Reed-Muller/Reed-Solomon codes and tensored BCH/repetition codes. The lengths of these codes $n_1 n_2$, are not prime. This introduces an amount of structure to the linear code that can be exploited by

an attacker, so the code is extended beyond size $(n_1 n_2)$ to the next primitive prime number n. Then, the last $l = n - n_1 n_2$ bits of the code can be truncated, when necessary. This introduces the following decoding problem, upon which the security of the system is based.

Definition 3.5.17 (Decoding with l erasures [48, p.12])**.** Let $\mathcal{C}[n,k]$ be a QC-code generated by $\mathbf{G}$ and $\mathbf{c} = \mathbf{mG} + \mathbf{e}$ for some random error $\mathbf{e} \xleftarrow{\$} \mathcal{S}_w^n(\mathbb{F}_2)$ ($\mathcal{S}_w^n$ is the set of length n vectors of weight w). Consider the matrix $\mathbf{G}' \in \mathbb{F}_2^{k \times n'}$ and error vector $\mathbf{e}' \in \mathbb{F}_2^{n'}$ obtained by removing the last $l = n - n' \geq 1$ columns from $\mathbf{G}$ and $\mathbf{e}$. The *Decoding with l erasures* problem asks to recover $\mathbf{m} \in \mathbb{F}_2^k$ from $\mathbf{c}' = \mathbf{mG}' + \mathbf{e}' \in \mathbb{F}_2^{n'}$ and $\mathbf{G}' \in \mathbb{F}_2^{k \times n'}$.

Since this is as decoding the encoded message given less information than in the 2-DQCSD and 3-DQCSD problems of Definitions 3.5.13 and 3.5.15 it is argued [48, p.12] that it is even harder than the original problems.

The security of the system is IND-CPA and is based on the assumption that the 2-DQCSD and 3-DQCSD problems of Definitions 3.5.13 and 3.5.15 with erasures are $\mathcal{NP}$-hard. There is a known algorithm for converting an IND-CPA system into an IND-CCA2 system but, due to decryption errors that would be introduced these cannot be used in HQC.

3.5.2 System Parameters

The parameters for HQC are given in chart form such that the workfactor involved in the best known attack on the algorithm is minimally above the desired security level.

Since HQC is specified for both BCH codes tensored with repetition codes as well as concatenated codes of Reed-Muller and Reed-Solomon there are two different tables.

For tensor product codes, the parameters suggested are as follows [48, p.37]:

Instance	n_1	n_2	n	k	δ	w	$w_r = w_e$	λ	p_{fail}
hqc-128	766	31	23,869	256	57	67	77	128	$< 2^{-128}$
hqc-192	766	59	45,197	256	57	101	117	192	$< 2^{-192}$
hqc-256	796	87	69,259	256	60	133	153	256	$< 2^{-256}$

In the tensor case, we are using shortened BCH$[n_1, k, \delta_1]$ codes and a repetition code $\mathbb{1}_{n_2}$ combined as per Definition 3.5.3 to produce a tensor product code $\mathcal{C} =$ BCH$[n_1, k_1, \delta] \otimes \mathbb{1}_{n_2}$.

The parameters are as follows:

- n_1 the length of the shortened BCH code,
- n_2 the length of the repetition code,
- n the smallest prime greater than the product $n_1 n_2$,
- k the length of the key being exchanged or message to be encrypted,
- δ the correcting capacity of the code,
- w the weight of the secret key,
- w_r, w_e the weight of the errors being introduced during encryption or encapsulation,
- λ the desired security level, and
- p_{fail} the probability of a decoding failure.

For the concatenated codes the parameters suggested are as follows.

Instance	n_1	n_2	n	w	$w_r = w_e$	λ	p_{fail}
hqc-RMRS-128	80	256	20,533	67	77	128	$< 2^{-128}$
hqc-RMRS-192	76	512	38,923	101	117	192	$< 2^{-128}$
hqc-RMRS-256	78	768	59,957	133	153	256	$< 2^{-128}$

The parameters are as follows:

- n_1 the length of the shortened BCH code,
- n_2 the length of the repetition code,
- n the smallest prime greater than the product $n_1 n_2$,

- k the length of the key being exchanged or message to be encrypted,
- δ the correcting capacity of the code,
- w the weight of the secret key,
- w_r, w_e the weight of the introduced errors,
- λ the desired security level, and
- p_{fail} the probability of a decoding failure.

3.5.3 Key Generation

During key generation, the above parameters are used to generate a secret and public key pairs as follows. The key generation for the KEM and the PKA are identical.

Algorithm 3.5.18 (HQC Key Generation [48, p.15])**.**

Input: (λ): The target security level.

Output: $(\mathbf{x}, \mathbf{y})$: The secret key.
$(\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$: The public key.

1: From the target security level λ, look up associated parameters $n, k, \delta, w, w_r, w_e$.
2: Sample $\mathbf{h} \xleftarrow{\$} \mathcal{R}$.
3: Sample the generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ of $\mathcal{C}$.
4: Generate sk $= (\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}$ such that $w(\mathbf{x}) = w(\mathbf{y}) = w$.
5: Set pk $= (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$.
6: **return** (pk, sk).

3.5.4 Encrypt/Encapsulate

The encryption and encapsulation operations are similar but the described security levels are different. The encryption algorithm sets up an IND-CPA system whereas the encapsulation system is fully IND-CCA2. For a fully IND-CCA2 system, the system must be indistinguishable to an attacker with access to a decryption oracle to which he can submit anything but the ciphertext. Several techniques exist that convert IND-CPA to IND-CCA2, but as in the case of BIKE many of these cannot

be used with a non-zero decoding failure rate. The Fujisaki-Okomoto transform can and is used to convert the IND-CPA Encryption to the IND-CCA2 KEM.

Algorithm 3.5.19 (HQC Encryption - IND-CPA [48, p.15])**.**

Input: $(\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$: The public key.
$(\mathbf{m})$ The message.

Output: $(\mathbf{c} = \mathbf{u} \cdot \mathbf{v})$: The encrypted message.

1: Generate $\mathbf{e} \xleftarrow{\$} \mathcal{R}$ such that $w(\mathbf{e}) = w_e$.
2: Generate $\mathbf{r} = (\mathbf{r_1}, \mathbf{r_2}) \xleftarrow{\$} \mathcal{R}^2$ such that $w(\mathbf{r}_1) = w(\mathbf{r}_2) = w_r$.
3: Set $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r_2} + \mathbf{e}$.
4: Return $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.

The encapsulation algorithm requires a hash function $\mathcal{G}$, and $\mathcal{H}$ this is typically a SHA or SHAKE hash. In the case of HQC, it is recommended that $\mathcal{G}$ uses SHA3-512 and $\mathcal{H}$ uses SHA-512. These are NIST standard hash functions and specifications can be found online.

Algorithm 3.5.20 (HQC Encapsulation - IND-CCA2 [48, p.15])**.**

Input: $(\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}))$: The public key.
(l): The length of the desired key.

Output: $(\mathbf{c} = \mathbf{u} \cdot \mathbf{v})$: The encapsulated key.

1: Generate $\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^l$ ▷ This seeds the desired key.
2: Derive the random seed $\theta \leftarrow \mathcal{G}(\mathbf{m})$.
3: Generate the ciphertext $\mathbf{c} \leftarrow (\mathbf{u}, \mathbf{v}) = \text{Encrypt}(pk, \mathbf{m}, \theta)$ ▷ (Algorithm 3.5.19).
4: Derive the symmetric key $k \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$.
5: Set $d \leftarrow \mathcal{H}(\mathbf{m})$
6: **return** $(\mathbf{c}, \mathbf{d})$.

This is again a pared-down version of the Fujisaki-Okomoto conversion with no message to be transferred. The only concern we have is sharing the symmetric key with the other system. The system generates a random message, uses a hash function to convert this to a seed, encrypts the message using the seed to produce an error vector and then uses this ciphertext and the random message $\mathbf{m}$ to produce the shared key. The message is hashed using a different hash algorithm and passed as part of the ciphertext to ensure that the value of $\mathbf{m}$ is decoded properly.

3.5.5 Decrypt/Decapsulate

Decryption in the PKE is a simple matter of decoding the received message.

Algorithm 3.5.21 (HQC Decryption - IND-CPA [48, p.15])**.**

Input: $(\mathbf{x}, \mathbf{y})$: The secret key.
$(\mathbf{c} = (\mathbf{u}, \mathbf{v}))$: The encrypted message .

Output: $(\mathbf{m})$: The message.

1: **return** $\mathbf{m} = \text{Decode}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$

The proper decode algorithm to use is determined by which flavour of HQC we are using, either concatenated or tensor codes.

Algorithm 3.5.22 (HQC Decapsulation - IND-CCA2 [48, p.15])**.**

Input: $(\mathbf{x}, \mathbf{y})$: The secret key.
(pk): The public key
$(\mathbf{c}, \mathbf{d})$: The encrypted key data.

Output: $(\mathbf{k})$: The shared secret key.

1: Decrypt $\mathbf{m}' \leftarrow \text{Decrypt}(sk, \mathbf{c})$ ▷ (Algorithm 3.5.22)
2: Compute $\theta' \leftarrow \mathcal{G}(\mathbf{m}')$
3: Reencrypt $\mathbf{c}' = \text{Encrypt}(pk, \mathbf{m}', \theta')$ ▷ (Algorithm 3.5.19) for verification.
4: **if** $\mathbf{c} \neq \mathbf{c}'$ or $\mathbf{d} \neq \mathcal{H}(\mathbf{m}')$ **then**
5: abort.
6: **end if**
7: **return** the shared key $\mathbf{k} \leftarrow \mathcal{K}(m, c)$

If the algorithm aborts, it should trigger the encapsulation algorithm to produce a new key and re-transmit.

3.5.6 Attacks on System

Apart from the ISD attacks that have been discussed in previous sections, attacks on HQC fall into two categories structural attacks and side-channel attacks. The best known attacks have a complexity of $2^{cw(1+\mathcal{O}(1))}$ for some constant c. An improvement by Sendrier [61] called "Decoding One of Many" (DOOM) should be considered. This is where the attacker has several ciphertexts but only needs to decode one of them. This translation of the problem produces a gain of $\mathbf{O}(\sqrt{n})$ over the standard problem.

The structural attacks are based on attacks on the form of the polynomial defining the ring $\mathcal{R}$. Such attacks have been studied by Guo, Johansson and Löndahl [34] for reducible polynomials, Löndahl et al. [43] for codes of even length and dimension and Sendrier [61] for reducing the search space when an attacker has access to multiple message but is satisfied decoding only one [61]. These attacks are thwarted by choosing $\mathcal{R}$ such that it only has two irreducible factors, which for good choices of n we have here since $\mathcal{R} = \mathbb{F}_2[x]/(x^n - 1)$ and n is prime.

As for side-channel attacks, it has been shown by Wafo-Tapa et al. [73] and Paiva and Terada [55] that HQC is prone to information leaks via timing differences unless the BCH decoding is done in constant time.

Security Analysis

Based on the above summary, the different sets of parameters that have been proposed in Section 3.5.2 have been calculated using the best known attacks to produce security recommended by NIST for $\lambda = 1, 3,$ or 5. That is of a computation complexity more than that of AES-128, AES-192 and AES-256, respectively.

The best known attacks against HQC have a complexity of $2^{-t \log(1-R)(1+o(1))}$ where $t = 2w,\ R = 1/2,\ o(1) = \log\left(\binom{n}{w})^2\right)/\left(\binom{2n}{2w}\right)$ for 2-DQCSD and $t = 3w,\ R = 1/3,\ o(1) = \log\left(\binom{n}{w_r}^3/\binom{3n}{3w_r}\right)$ for 3-DQCSD. To account for the DOOM attack, this factor should be divided by $\sqrt{n}$ [48, p.49].

Chapter 4

Conclusion and Future Work

This thesis is intended as a survey of current methods being used in this field and to establish some problems which may pose interesting for further research. Coding theory is introduced in order to provide the necessary background for understanding the four different types of key encapsulation methods that are described.

From a security standpoint, we see that of the four key encapsulation methods, attack techniques take one of three different forms side-channel attacks, mathematical attacks on the structure, and Information Set Decoding (ISD) attacks. The ISD attacks are susceptible to a $\sqrt{n}$ reduction on a quantum computer based on Grover's quantum search algorithm.

Side-channel attacks are any attacks that provide information about the messages or the keys by some information leaking out related to the running of the algorithms on computer systems and not through weaknesses in their mathematical foundations.

Timing attacks are side-channel attacks that use the time of encoding or decoding messages and knowledge of the underlying algorithms to recover information about the keys being produced. In the case of BIKE, LEDAcrypt the bit-flipping decoding mechanisms run in constant time in an effort to discourage such attacks. In earlier BIKE submissions, the bit-flip algorithm did not run in constant time, but this was updated in May 2020 to ensure that it did. Classic McEliece Key Encapsulation implementations ensure that decapsulation does not immediately abort on finding an error token. HQC requires that BCH be run in constant time in order to avoid information leaks through timing differences.

Another avenue of side-channel information leakage is through a power-difference analysis. We have not seen any work in this area for any of the above systems so as future work, this area seems interesting.

Algebraic attacks attempt to use some of the underlying structure of the error correcting codes to short-cut attacks on the system. In over 40 years since its introduction, the underlying algebraic structures of Goppa codes have not seen any successful attacks. The attack of Löndahl et al. [43] forces BIKE, LEDAcrypt, and HQC to select the underlying group so that it has no secondary factors (i.e. $\mathcal{R} = \mathbb{F}_2[x]/(x^r + 1)$ where $r - 1$ is prime). The most efficient brute-force type attacks take the form of ISD attacks and quantum ISD attacks. The parameters chosen for all four of the cryptosystems studied use this form of attack as the motivating factor for parameter choices.

From this survey of the current NIST candidates for code-based key encapsulation mechanisms, the following research problems arise:

- A power-differential analysis of the implementations of all four cryptosystems could be performed to find any potential information leaks.
- The root finding in Patterson's algorithm (Algorithm 2.3.8) could be improved by using modern polynomial factorization techniques. It would be interesting to see which algorithms are in use in Classic McEliece and determine if these can be improved upon.
- There are no digital signature methods using error correcting codes. This could be a very interesting design challenge. Pioneering work in this direction has been published in August 2020 by Baldi et al. [7]
- Cryptosystems designed using error correcting codes based on other metrics such as Lee or poset metric.
- Study other improvements to ISD decoding.
- BIKE and LEDAcrypt are using different versions of the bit-flipping algorithm. Attempt to improve upon these bit-flipping methods.

In July of 2020, NIST announced the Round 3 candidates for post-quantum cryptography. The finalists for KEM/Encryption systems and digital signature systems are outlined in the following tables.

Public-key Encryption and Key-establishment Algorithms - Finalists

Classic McEliece	Code-Based
CRYSTALS-KYBER	Lattice
NTRU	Lattice-Based
Saber	Lattice-Based

Digital Signature Algorithms - Finalists

CRYSTALS-KYBER	Lattice
FALCON	Lattice
Rainbow	Multivariate

LEDAcrypt has been removed from consideration. This is likely due to the combination of a few factors. The weak-key attack of April 2020 [2] reduced confidence in this cryptographic system as the specification had to be rewritten to strengthen the keys against this. In addition, there were two competing candidates based on bit-flipping algorithms with LDCP/MDCP codes and as duplication in the standards is not desirable, the apparent weakness of LEDAcrypt forced its removal from consideration.

To the matter of the survival chances of Classic McEliece in the competition there are a few matters of note. Firstly, there is no competition between multivariate and code-based cryptosystems as there is no overlap in their uses. Currently, code-based cryptography is used for key-establishment and encryption whereas multivariate cryptography is used for digital signatures. The competition for both comes from lattice-based systems. These lattice-based systems boast smaller key sizes than multivariate or code-based system and in the case of CRYSTALS, there is some overlap between the digital signature and key exchange mechanisms that reduce the cost of implementing a system that required both operations.

A key advantage of code-based and multivariate system lies in the fact that they are suitably different from lattice based system and as such stand a better chance of surviving, depending on how far NIST wants to go in reducing the duplication of standards.

List of References

[1] N. Alon and J. H. Spencer, *The Probabilistic Method*. John Wiley & Sons, 2016.

[2] D. Apon, R. A. Perlner, A. Robinson, and P. Santini, "Cryptanalysis of LEDAcrypt," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 455, 2020.

[3] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. Zémor, "BIKE: Bit Flipping Key Encapsulation Round 3 Submission." Available online: `https://bikesuite.org/`, 2020. Accessed on 17 February 2020.

[4] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, and G. Zémor, "BIKE: Bit Flipping Key Encapsulation Round 1 Submission." Available online: `https://bikesuite.org/`, 2017. Accessed on 5 July 2020.

[5] N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. Zémor, "BIKE: Bit Flipping Key Encapsulation Round 3 Submission." Available online: `https://bikesuite.org/files/v4.0/BIKE_Spec.2020.05.03.1.pdf`, 2020. Accessed on 23 May 2020.

[6] M. Baldi, *QC-LDPC Code-based Cryptography*. Springer Science & Business, 2014.

[7] M. Baldi, M. Battaglioni, F. Chiaraluce, A.-L. Horlemann-Trautmann, E. Persichetti, P. Santini, and V. Weger, "A New Path to Code-based Signatures via Identification Schemes with Restricted Errors." Available online: `https://arxiv.org/pdf/2008.06403.pdf`, August 17, 2020. Accessed on August 29, 2020.

[8] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "LEDAcrypt: Low-dEnsity parity-check coDe-bAsed cryptographic systems Specification revision 2.5 March, 2020." Available online: `https://www.ledacrypt.org/documents/LEDAcrypt_spec_2_5.pdf`, 2020. Accessed on 14 April 2020.

[9] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "LEDAcrypt: Low-dEnsity parity-check coDe-bAsed cryptographic systems Specification revision 3 April, 2020." Available online: `https://www.ledacrypt.org/documents/LEDAcrypt_spec_2_5.pdf`, 2020. Accessed on 09-July-2020.

[10] E. Barker and J. Kelsey, "NIST Special Publication 800-90A (A Revision of SP 800-90) Recommendation for Random Number Generation Using Deterministic Random Bit Generators," 2012.

[11] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, "Relations Among Notions of Security for Public-key Encryption Schemes," in *Annual International Cryptology Conference (CRYPTO)*, vol. 1462 of Lecture Notes in Computer Science, pp. 26–45, Springer, 1998.

[12] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the Inherent Intractability of Certain Coding Problems," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[13] E. Berlekamp, "Goppa Codes," *IEEE Transactions on Information Theory*, vol. 19, no. 5, pp. 590–592, 1973.

[14] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the Inherent Intractability of Certain Coding Problems," *IEEE Transactions on Information Theory*, vol. 24, no. 3, pp. 384–386, 1978.

[15] E. R. Berlekamp, *Algebraic Coding Theory, revised ed.* World Scientific Publishing Co., 2015.

[16] D. J. Bernstein, "Grover vs. McEliece," in *International Workshop on Post-Quantum Cryptography (PQCRYPTO)*, vol. 6061 of Lecture Notes in Computer Science, pp. 73–80, Springer, 2010.

[17] D. J. Bernstein, T. Chou, T. Lange, R. Misoczki, R. Niederhagen, E. Persichetti, P. Schwabe, J. Szefer, and W. Wang, "Classic McEliece: Conservative Code-based Cryptography 30 March 2019." `https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/`

documents/round-2/submissions/classic-mceliece-round2.zip, 2019. Retrieved March 4, 2020.

[18] D. J. Bernstein and B.-Y. Yang, "Fast Constant-time GCD Computation and Modular Inversion," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 340–398, 2019.

[19] J. A. Bondy and U. S. R. Murty, "Graph Theory," *volume 244 of Graduate texts in Mathematics*, p. 81, 2008.

[20] R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Information and Control*, vol. 3, no. 1, pp. 68–79, 1960.

[21] A. Canteaut and F. Chabaud, "A New Algorithm for Finding Minimum-weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-sense BCH Codes of Length 511," *IEEE Transactions on Information Theory*, vol. 44, no. 1, pp. 367–378, 1998.

[22] J. Chaulet, *Étude de cryptosystèmes à clé publique basés sur les codes MDPC quasi-cycliques*. PhD thesis, Paris 6, 2017.

[23] W. Diffie and M. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[24] N. Drucker, S. Gueron, and D. Kostic, "On Constant-time QC-MDPC Decoding with Negligible Failure Rate," Technical Report, Cryptology ePrint Archive, Report 2019/1289, 2019.

[25] N. Drucker, S. Gueron, and D. Kostic, "Fast Polynomial Inversion for Post Quantum QC-MDPC Cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 298, 2020.

[26] N. Drucker, S. Gueron, and D. Kostic, "QC-MDPC Decoders with Several Shades of Gray," in *International Conference on Post-Quantum Cryptography*, vol. 12100 of Lecture Notes in Computer Science, pp. 35–50, Springer, 2020.

[27] N. Drucker, S. Gueron, D. Kostic, and E. Persichetti, "On the Applicability of the Fujisaki-Okamoto Transformation to the BIKE KEM," Technical Report, IACR Cryptology ePrint Archive, 2020.

[28] D. Engelbert, R. Overbeck, and A. Schmidt, "A Summary of McEliece-type Cryptosystems and their Security," *Journal of Mathematical Cryptology*, vol. 1, no. 2, pp. 151–199, 2007.

[29] E. Fujisaki and T. Okamoto, "Secure Integration of Asymmetric and Symmetric Encryption Schemes," in *Annual International Cryptology Conference (CRYPTO)*, vol. 1666 of Lecture Notes in Computer Science, pp. 537–554, Springer, 1999.

[30] R. Gallager, "Low-density Parity-check Codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[31] E. N. Gilbert, "A Comparison of Signalling Alphabets," *The Bell system technical journal*, vol. 31, no. 3, pp. 504–522, 1952.

[32] V. D. Goppa, "A New Class of Linear Correcting Codes," *Problemy Peredachi Informatsii*, vol. 6, no. 3, pp. 24–30, 1970.

[33] L. K. Grover, "A Fast Quantum Mechanical Algorithm for Database Search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 212–219, 1996.

[34] Q. Guo, T. Johansson, and C. Löndahl, "A New Algorithm for Solving Ring-LPN with a Reducible Polynomial," *IEEE Transactions on Information Theory*, vol. 61, no. 11, pp. 6204–6212, 2015.

[35] Q. Guo, T. Johansson, and P. Stankovski, "A Key Recovery Attack on MDPC with CCA Security Using Decoding Errors," in *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, vol. 10031 of Lecture Notes in Computer Science, pp. 789–815, Springer, 2016.

[36] R. W. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[37] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, no. 2, pp. 147–56, 1959.

[38] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in GF (2^m) Using Normal Bases," *Information and Computation*, vol. 78, no. 3, pp. 171–177, 1988.

[39] P. J. Lee and E. F. Brickell, "An Observation on the Security of McEliece's Public-key Cryptosystem," in *Workshop on the Theory and Application of of Cryptographic Techniques*, pp. 275–280, Springer, 1988.

[40] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and their Applications*. Cambridge University Press, 1986.

[41] R. Lidl and H. Niederreiter, *Finite Fields*, vol. 20. Cambridge University Press, 1997.

[42] Z. Liu, Y. Pan, and T. Xie, "Breaking the Hardness Assumption and IND-CPA Security of HQC Submitted to NIST PQC Project," *IET Information Security*, vol. 14, no. 3, pp. 313–320, 2019.

[43] C. Löndahl, T. Johansson, M. K. Shooshtari, M. Ahmadian-Attari, and M. R. Aref, "Squaring Attacks on McEliece Public-key Cryptosystems Using Quasi-cyclic Codes of Even Dimension," *Designs, Codes and Cryptography*, vol. 80, no. 2, pp. 359–377, 2016.

[44] D. J. MacKay and R. M. Neal, "Good Codes Based on Very Sparse Matrices," in *IMA International Conference on Cryptography and Coding*, pp. 100–111, Springer, 1995.

[45] D. J. MacKay and R. M. Neal, "Near Shannon Limit Performance of Low Density Parity Check Codes," *Electronics letters*, vol. 32, no. 18, pp. 1645–1646, 1996.

[46] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, vol. 16. Elsevier, 1977.

[47] R. J. McEliece, "A Public-key Cryptosystem Based on Algebraic," *Coding Thv*, vol. 4244, pp. 114–116, 1978.

[48] C. A. Melchor, N. Aragon, S. Bettaieb, L. Bidoux, O. Blazy, J. Bos, J.-C. Deneuville, P. Gaborit, E. Persichetti, J.-M. Robert, P. Véron, and G. Zémor, "Hamming Quasi-Cyclic (HQC)," Technical Report, National Institute of Standards and Technology 2017, 2020.

[49] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. Barreto, "MDPC-McEliece: New McEliece Variants from Moderate Density Parity-check Codes," in *2013 IEEE International Symposium on Information Theory*, pp. 2069–2073, IEEE, 2013.

[50] G. L. Mullen and D. Panario, *Handbook of Finite Fields*. Chapman and Hall/CRC, 2013.

[51] H. Niederreiter, "Knapsack-type Cryptosystems and Algebraic Coding Theory," *Probl. Control and Inform. Theory*, vol. 15, pp. 159–166, 1986.

[52] National Institute of Standards and Technology, "Post-Quantum Cryptography (PQC) - Round 2 Submissions." https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions, 2020. Retrieved June 9, 2020.

[53] M. Albrecht, C. Cid, K. Paterson, C. Tjhai, and M. Tomlinson, "NTS-KEM, 2019." https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions, 2020. Retrieved June 9, 2020.

[54] R. Overbeck, *Public Key Cryptography Based on Coding Theory*. PhD thesis, Technische Universität, 2007. Retrieved March 3, 2020.

[55] T. B. Paiva and R. Terada, "A Timing Attack on the HQC Encryption Scheme," in *International Conference on Selected Areas in Cryptography (SAC2019)*, vol. 11959 of Lecture Notes in Computer Science, pp. 551–573, Springer, 2019.

[56] N. Patterson, "The Algebraic Decoding of Goppa Codes," *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 203–207, 1975.

[57] E. Prange, "The Use of Information Sets in Decoding Cyclic Codes," *IRE Transactions on Information Theory*, vol. 8, no. 5, pp. 5–9, 1962.

[58] C. C. Pugh, *Real Mathematical Analysis*, vol. 2011. Springer, 2002.

[59] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[60] N. Sendrier, "Code-based Cryptography." https://2017.pqcrypto.org/school/slides/cbctutorial.pdf, 2017. Retrieved April 22, 2019.

[61] N. Sendrier, "Decoding One Out of Many," in *International Workshop on Post-Quantum Cryptography (PQCRYPTO)*, vol. 7071 of Lecture Notes in Computer Science, pp. 51–67, Springer, 2011.

[62] P. W. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," in *Proceedings 35th annual symposium on foundations of computer science*, pp. 124–134, IEEE, 1994.

[63] T. Shrimpton, "A Characterization of Authenticated-Encryption as a Form of Chosen-Ciphertext Security," *IACR Cryptology ePrint Archive*, p. 272, 2004.

[64] V. M. Sidelnikov and S. O. Shestakov, "On Insecurity of Cryptosystems Based on Generalized Reed-Solomon Codes," *Discrete Mathematics and Applications*, vol. 2, no. 4, pp. 439–444, 1992.

[65] M. Sipser and D. A. Spielman, "Expander Codes," *IEEE transactions on Information Theory*, vol. 42, no. 6, pp. 1710–1722, 1996.

[66] J. Stern, "A Method for Finding Codewords of Small Weight," in *International Colloquium on Coding Theory and Applications*, pp. 106–113, Springer, 1988.

[67] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A Method for Solving Key Equation for Decoding Goppa Codes," *Information and Control*, vol. 27, no. 1, pp. 87–99, 1975.

[68] J.-P. Tillich, "The Decoding Failure Probability of MDPC Codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*, pp. 941–945, IEEE, 2018.

[69] M. Tomlinson, C. J. Tjhai, M. A. Ambroze, M. Ahmed, and M. Jibril, *Error-Correction Coding and Decoding*. Springer, 2017.

[70] R. C. Torres and N. Sendrier, "Analysis of Information Set Decoding for a Sublinear Error Weight," in *Post-Quantum Cryptography (PQCRYPTO)*, vol. 9606 of Lecture Notes in Computer Science, pp. 144–161, Springer, 2016.

[71] R. R. Varshamov, "Estimate of the Number of Signals in Error Correcting Codes," *Docklady Akad. Nauk, SSSR*, vol. 117, pp. 739–741, 1957.

[72] J. von zur Gathen and D. Panario, "Factoring Polynomials Over Finite Fields: A Survey," *Journal of Symbolic Computation*, vol. 31, no. 1-2, pp. 3–17, 2001.

[73] G. Wafo-Tapa, S. Bettaieb, L. Bidoux, and P. Gaborit, "A Practicable Timing Attack Against HQC and its Countermeasure," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 909, 2019.