M.Sc. Thesis
Master of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Code-Based Post-Quantum Cryptography

Hamming Quasi-Cyclic Encryption Scheme

Amalie Due Jensen (s160503)

Kongens Lyngby 2022

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

Public-key cryptography plays an important role in modern communication systems. It is well known that the security of todays' most used public-key encryption algorithms such as RSA and ElGamal is vulnerable in the presence of quantum computers due to e.g. Shor's quantum algorithm from 1995. Therefore, NIST started a world-wide competition-like standardization process of new public-key quantum-resistant algorithms in 2017. Today, the 82 submissions have been reduced to 7 finalists and 8 alternates. The code-based cryptosystem, HQC, is among the 8 alternates and is a quite new cryptosystem (published in 2017). Compared to the old Classic McEliece from 1978 among the 7 finalists, the public key size of HQC is much smaller which is a great advantage. HQC uses two codes, a publicly known code $\mathcal{C}$ and a random code, and the security of HQC relies on the hardness of decoding the random code. Initially, the code $\mathcal{C}$ was a product code of a shortened BCH code and a repetition code, but it was later replaced by a concatenated code of a shortened Reed-Solomon code and a duplicated Reed-Muller code. This thesis aims to investigate if an even better choice of $\mathcal{C}$ exists. Four error-correcting codes have been implemented in SageMath, and different concatenated codes, product codes, and a couple of single codes have been designed using combinations of the chosen set of codes. Simulations have been run on HPC cluster hardware to measure the performance of each designed code as $\mathcal{C}$ in HQC. The best performing new code was a product code of an $\mathcal{RM}(r = 4, m = 9)$ code and a repetition code of length $n = 33$. This code seems promising as a candidate for HQC as it allows for reducing the public key size even more while keeping acceptable security level estimates. Additionally, simulations have shown that if it is possible for an attacker to compromise the system and force the parameter $\mathbf{h}$ representing the parity-check matrix of the random code, to be either very close to 0 or very close to $n_1 n_2$, then the cryptosystem would be significantly easier to break with e.g. fault attacks.

Preface

This 32.5 ECTS points master thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a master's degree in Mathematical Modelling and Computing. The work was conducted under supervision of Assistant Professor Tyge Tiessen and Ph.D. student Freja Elbro at the Technical University of Denmark as well as Professor Dr.-Ing. Antonia Wachter-Zeh and Ph.D. student Sabine Pircher at the Technical University of Munich.

Kongens Lyngby, January 17, 2022

Amalie Due Jensen (s160503)

Acknowledgements

I would like to acknowledge the assistance of Tyge Tiessen and Freja Elbro who have both been supportive and provided me with encouragement throughout the duration of the project. Tyge has been helpful with administrative planning, and Freja's help with algebra has been extremely valuable.

I am really grateful to Antonia Wachter-Zeh and Sabine Pircher for a lot of support from Munich. Already from day one, Antonia has been very open minded, and I would like to express my deepest appreciation to Antonia for making it possible for me to stay in Munich for 6 weeks during the process. The help from Sabine cannot be overstimated. Sabine has played a decisive role throughout the whole process. She gave me a lot of support and practical advice during my visit in Munich. Furthermore, the collaboration regarding HQC with her has been really valuable and has also been a lot of fun. Many thanks also to Violetta Weger and Georg Maringer and the rest of Antonia's group for giving me an amazing time in Munich.

I would like to thank Sven Puchinger for establishing the contact between me and Antonia and for being helpful already 2 years ago when we got in touch.

I would like to thank the DTU DCC resources and Associate Professor Bernd Dammann for making it possible to run simulations using HPC cluster hardware [DTU21].

Finally, I am extremely grateful to my boyfriend, family, and the friends at DTU who have been supportive since we met in 2016. Thanks a lot!

Contents

CHAPTER 1

Introduction

During the last decades, the need for quantum-resistant public-key encryption schemes has been identified. The general idea in a public-key encryption scheme is to construct a one-way function, i.e. encryption should be easy, but decryption without the secret key should be hard. The most used approaches are to make the security of the encryption scheme rely on either the factoring problem or the discrete logarithm problem. These two mathematical problems are both easy to set up (encryption is easy), but hard to solve (decryption without secret key is NP-hard), and therefore they are suitable as one-way functions. Unfortunately, with the presence of quantum-computers, these two mathematical problems are no longer NP-hard as polynomial-time quantum algorithms that solve these mathematical problems have already been developed. An example is Shor's algorithm which is a polynomial-time quantum algorithm for the factoring problem, developed by Peter Shor in 1995 [IBMb]. Given that the security of our daily online transactions relies on the assumption that the factoring problem is NP-hard, it is catastrophical if this assumption does not hold anymore, and therefore quantum-resistant cryptography is needed.

Luckily, a lot of research within post-quantum cryptography, i.e. quantum-resistant cryptography, is already being done around the world. Already in April 2015, the National Institute of Standards and Technology (NIST) held a Post Quantum Workshop to engage academic, industry, and government stakeholders to prioritize the research within the field [NIS21]. In February 2016, a PQCrypto conference was held, also by NIST, in Japan where NIST announced a call for public-key quantum-resistant encryption schemes and that they would begin a worldwide competition-like standardization process of new public-key quantum-resistant encryption schemes. Different classes of post-quantum encryption schemes are being considered. This includes *lattice-based*, *code-based*, and *hash-based* encryption schemes, but the goal is common for all approaches: To find a difficult quantum-resistant mathematical problem that the security of the encryption scheme can rely on. The timeline of the NIST standardization process is shown below [20221]:

- November 2017: Submission deadline. NIST received 23 signature schemes and 59 KEM/encryption schemes totalling 82 schemes. The different scheme types are explained in Chapter 2.

- December 2017 - January 2019: The first round of the standardization process was held. In the end, the 26 most promising schemes were selected to move on.
- January 2019 - July 2020: The second round of the standardization process was held. In the end, 15 schemes were selected to move on.
- July 2020 - now: The third round is still being held. NIST has identified 7 finalists and 8 alternates, i.e. candidates for potential standardization after further research.

As NIST announced that they do not expect to pick out one winner, but instead it is preferrable to identify several good algorithms, it is still highly relevant to also consider the alternates [20221]. One of the 8 alternates is a code-based encryption scheme called the *Hamming-Quasi Cyclic (HQC)* encryption scheme, and this encryption scheme will be the focus of this thesis. Code-based encryption schemes are based on *error-correcting codes*, and the NP-hard problem that the security relies on, partly or completely, for such a scheme is *decoding of a random code*. HQC uses two codes, a code denoted $\mathcal{C}$ and a random code. If the random code can be decoded, then HQC is broken. Unlike most of the existing code-based encryption schemes such as Classic McEliece from 1978 among the 7 finalists, the code $\mathcal{C}$ in HQC is public.

The concept of error-correcting codes is not a part of the field of cryptography, instead it lies within the algebraic field. An error-correcting code consists of a set of *codewords*, and when such a codeword is sent over a noisy channel, the receiver on the other side receives an erronouos codeword. The task of the error-correcting code is then to reconstruct the original codeword which is possible as long as not too many errors occurred during the transmission. In the first version of HQC, the error-correcting code $\mathcal{C}$ was a product code constructed using a shortened BCH code and a repetition code. A product code is a way of deriving a new code by combining two existing codes. In later versions, the product code was replaced by a concatenated code constructed using a shortened Reed-Solomon code and a duplicated Reed-Muller code where a concatenated code is another way of deriving a new code. It is still an open question whether an even better choice of $\mathcal{C}$ for HQC can be found. A question which this thesis aims to investigate. Note that a code $\mathcal{C}$ is a better choice for HQC if it allows for reducing the public key size while still keeping the encryption scheme secure.

The goal of this thesis is to implement different error-correcting codes in the python library SageMath. Furthermore, different product codes, concatenated codes, as well as some single codes will be designed using combinations of the error-correcting codes that are covered. Furthermore simulations on HPC cluster hardware of HQC with different choices of $\mathcal{C}$ will be run to measure how each choice of $\mathcal{C}$ affects the security of HQC. Finally, the simulation results will be compared and discussed, and it will be discussed if any new insights in the encryption scheme has been identified throughout the experiments.

CHAPTER 2

Cryptography

In this Chapter, different concepts within the field of cryptography will be presented. This includes the basic encryption models and the concept of post-quantum cryptography.

2.1 Basic Encryption Models

In general, there are two types of encryption models, *symmetric* and *asymmetric* or *public-key* encryption models. These are presented in the following sections.

2.1.1 Symmetric Encryption Model

In a symmetric encryption model, only one key is used. The encryption and decryption tasks are both performed with the same key which means that the key is shared between the sender and the receiver as shown on Figure 2.1. An example of a symmetric cryptosystem is the *Advanced Encryption Standard (AES)*. Symmetric cryptography can be attacked by quantum computers using Grover's algorithm [IBMa]. However, it has been shown that resistance of symmetric cryptography can be achieved by increasing the key sizes [WP21, p. 7].

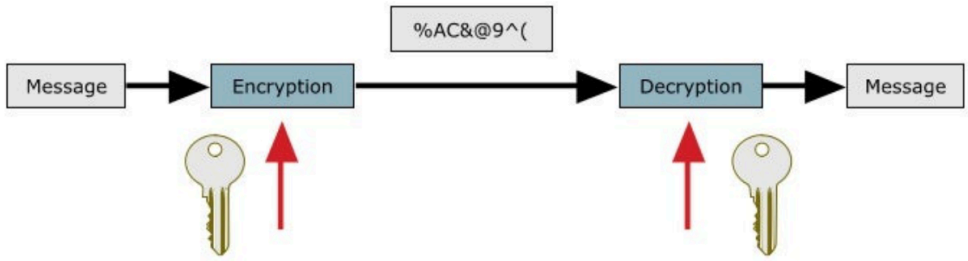

Figure 2.1: Symmetric encryption model. The same key is used for both encryption and decryption. The figure is borrowed from [Knu18b, p. 26].

2.1.2 Public-Key Encryption Model

In asymmetric or public-key encryption models, two keys are used, a *public key* for encryption and a *private key*, also called a *secret key*, for decryption. The sender uses the public key to encrypt, and only the person who has the secret key can decrypt as shown in Figure 2.2.

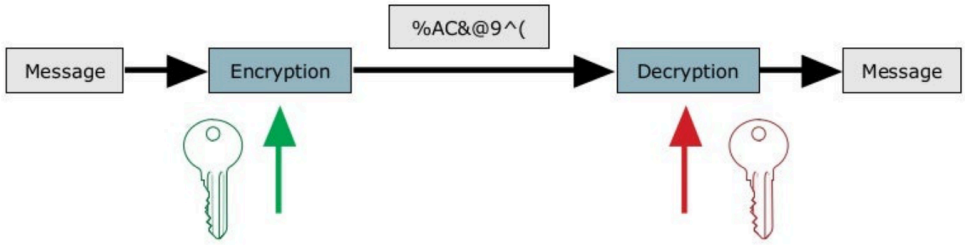

Figure 2.2: Asymmetric encryption model. The green key used for encryption is the public key, and the red key used for decryption is the secret key. The figure is borrowed from [Knu18b, p. 46].

When building a public-key encryption model, the idea is to build a one-way function, i.e. encryption should be easy, but decryption without the private key should be difficult. The two strategies that are mostly used today is to build public-key encryption models that rely on one of the two hard problems [Knu18b, p. 45]:

- The factoring problem: Given $n = pq$, find the two primes p and q.
- The discrete logarithm problem: Given a, n, and $b = a^x \mod n$, find x.

The most famous example of a public-key encryption model that relies on the factoring problem is RSA, and some famous examples of public-key encryption models that rely on the discrete logarithm problem are ElGamal and elliptic-curve cryptography.

2.2 Post-Quantum Cryptography

It is well known that quantum computers will be able to break public-key cryptosystems whose security relies on the factoring problem or the discrete logarithm problem, and therefore the term *post-quantum cryptography* has been introduced to describe public-key cryptosystems that will remain secure in the presence of quantum computers [Knu18a, p. 107].

Shor's quantum algorithm is a polynomial-time quantum algorithm that can solve the factoring problem faster than classical factoring algorithms. It achieves this by leveraging that quantum computers are able to determine a period in a given function efficiently. A function f is called periodic with period $t > 0$ if it holds that

$$f(x + t) = f(x),$$

for all possible values of x. Knowing the period allows for finding small multiples of $\phi(n)$ which makes it possible to factor n [Knu18a, p. 107]. As usual, $\phi(n)$ denotes Euler's phi function.

A variant of Shor's algorithm can be used to break the discrete logarithm problem.

The basic requirements for post-quantum public-key encryption algorithms are [WP21, p.9]:

- They should be based on NP-hard problems that are not breakable by polynomial attacks on quantum computers.
- They should be efficiently implementable.

2.3 Public-Key Encryption Scheme (PKE)

Throughout the thesis, the focus will be on the public-key encryption model as this is the one that is threatened by quantum computers. In this Section, the structure of the public-key encryption model will be investigated.

In general, the public-key encryption model consists of three algorithms [Agu+21a, p. 14]:

- Key generation: This algorithm takes as input the necessary parameters defining the model and outputs a pair of keys, the public key pk that will be used for encryption and the secret key sk that will be used for decryption.
- Encryption: This algorithm takes as input the message $\mathbf{m}$ and the public key pk and outputs the ciphertext $\mathbf{c}$.
- Decryption: This algorithm takes as input the ciphertext $\mathbf{c}$ and the secret key sk and outputs the message $\mathbf{m}$.

The general requirements to such a cryptosystem are both of the two following properties [Agu+21a, p. 14]:

- Correctness: The probability of retrieving the correct message **m** from decryption should be 1 or at least very close to 1 for any instance.
- Indistinguishability under Chosen Plaintext Attack (IND-CPA): An adversary should not be able to efficiently guess which plaintext has been encrypted even if he knows it is one of two different plaintexts of his own choice.

2.4 PKE, KEM, and Signature Schemes

As mentioned, NIST received in total 82 submissions to the standardization process of which 23 submissions were *signature schemes* and the remaining 59 submissions were *Key Encapsulation Mechanism (KEM)* schemes [20221]. Both types are techniques within public-key cryptography, and the difference is explained in the following way:

- A signature scheme works in the following way [Knu18b, p. 95]: Alice creates a key pair, a public key and a secret key. She publishes the public key and keeps the secret key private. She can then use the secret key to sign a message, i.e. she creates a signature. When Bob receives the message together with the signature, he can then use Alice's public key to verify that Alice created the signature and that the signature belongs to the message.
- A KEM is an encryption technique where a public-key encryption scheme (PKE) is used to exchange a symmetric key, e.g. an AES-key. In general, it is most efficient to use the symmetric key for encryption and only use the PKE for (symmetric) key exchange as symmetric encryption is much faster than public-key encryption.

HQC belongs to the KEM category. The version of HQC that will presented in this thesis is a PKE which is IND-CPA secure as described in Section 2.3. However, the PKE version of HQC can be transformed to a KEM which is *indistinguishable against adaptive chosen-ciphertext attacks (IND-CCA2)* which is the highest standard security requirement for a public-key cryptosystem [Agu+21a, p. 15], but this transformation will not be covered in this thesis.

CHAPTER 3

Coding Theory

Among the category of post-quantum public-key cryptosystems are the *code-based cryptosystems*. Code-based cryptosystems use error-correcting codes with the mathematically NP-hard problem that the security relies on being the decoding of a random code. In this chapter, the general concepts in coding theory and some different classes of error-correcting codes will be introduced.

3.1 Prerequisites

It is assumed that the reader is already familiar with the definitions of groups, rings, and prime fields. Throughout this thesis, let $\mathbb{F}_q$ denote the prime field of q elements for a prime q, and let $\mathbb{F}_{q^m}$ denote the extension field of the base field $\mathbb{F}_q$ where $\mathbb{F}_q$ is a prime field, and the integer $m \geq 1$ is called the *extension degree* [Rot06, p. 57]. Furthermore, let $\mathbb{F}_q^n$ denote a row vector of length n with elements from the prime field $\mathbb{F}_q$. Let $\mathbb{F}_q[x]$ denote the polynomial ring containing all polynomials in the variable x with coefficients in the field $\mathbb{F}_q$. Let $\mathbb{F}_q[X]/(f(x))$ denote all polynomials over $\mathbb{F}_q$ of degree less than the degree of $f(x)$ as they are reduced modulo $f(x)$, for a polynomial $f(x) \in \mathbb{F}_q[x]$.

Furthermore, recall that a vector space over a field $\mathbb{F}_q$ is a set $\mathcal{V}$ together with two operations, $+$ and $\cdot$, that satisfies a list of axioms [Wac+, p. 36]. Furthermore, a non-empty subset of vectors $\mathcal{U} \subseteq \mathcal{V}$ is itself a vector space and is called *subspace* if $\mathbf{u}, \mathbf{v} \in \mathcal{U}$ implies that

$$a \cdot \mathbf{u} + b \cdot \mathbf{v} \in \mathcal{U},$$

for all $a, b \in \mathbb{F}_q$.

A *basis* is a set of vectors that spans the whole vector space and is linearly independent. The *dimension* of $\mathcal{V}$ is the number of vectors in the basis.

A term that the reader is not expected to know beforehand is a *splitting field* which is defined in Definition 3.2. The Definition relies on the following Theorem 3.1 which is stated without a proof. A proof can be found in [Rot06, p. 64].

Theorem 3.1. *[Rot06, p. 64] Let $a(x)$ be a polynomial of degree $n \geq 0$ over a field $\mathbb{F}_q$. Then there exists an extension field $\mathbb{F}_{q^m}$ for some m in which $a(x)$ has n roots (counting multiplicity).*

Definition 3.2 (Splitting Field)**.** *[Rot06, p. 65] Given a field $\mathbb{F}_q$ and a polynomial $a(x) \in \mathbb{F}_q[x]$, an extension field $\mathbb{F}_{q^m}$ that satisifies the conditions in Theorem 3.1 with the smallest possible extension degree m is called a splitting field of $a(x)$ over $\mathbb{F}_q$.*

Finally, the terms *Hamming weight* and *Hamming distance* will be introduced:

Definition 3.3 (Hamming Weight)**.** *[Rot06, p. 6] Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a vector. The Hamming weight of $\mathbf{v}$, denoted $wt(\mathbf{v})$, is defined as the number of non-zero entries,*

$$wt(\mathbf{v}) := |\{i \mid v_i \neq 0, i = 0, \ldots, n-1\}|.$$

Definition 3.4 (Hamming Distance)**.** *[Rot06, p. 6] Let $\mathbf{v}$ and $\mathbf{w}$ be two vectors, both of length n. The Hamming distance between $\mathbf{v}$ and $\mathbf{w}$, denoted $d_H(\mathbf{v}, \mathbf{w})$, is the number of entries where they are different,*

$$d_H(\mathbf{v}, \mathbf{w}) := |\{i \mid v_i \neq w_i, i = 0, \ldots, n-1\}|.$$

It follows that the Hamming distance between two vectors corresponds to the Hamming weight of the difference of the vectors [Rot06, p. 6]:

$$d_H(\mathbf{v}, \mathbf{w}) = wt(\mathbf{v} - \mathbf{w}).$$

3.2 Introduction to Coding Theory

Coding theory is a mathematical discipline where codes with specific properties and structures are used for purposes such as communication and storage of information. An important term in coding theory is an *error-correcting code* which is a set of *codewords*. A codeword is a vector, also called a word, of some kind of symbols i.e. bits of a given length. When such a codeword is transmitted over a noisy channel, the receiver receives a codeword but in which some symbols have been changed due to noise. The task of the error-correcting code is then to reconstruct the original codeword. The transmission model is shown in Figure 3.1.

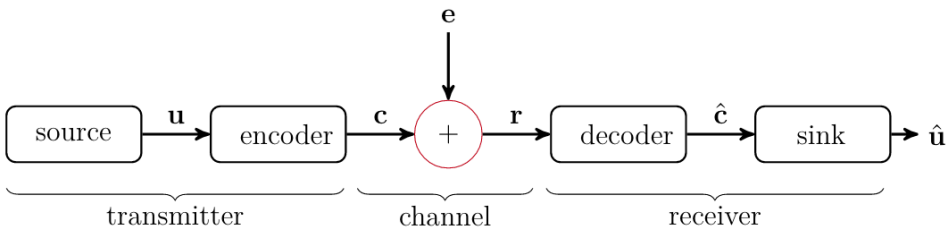

Figure 3.1: The general transmission model borrowed from[Wac+, p. 5].

The different elements on Figure 3.1 are explained below [Wac+, p. 5-6]:

- The *source* is where the information vector $\mathbf{u} = (u_0, \ldots, u_{k-1})$ with elements from a finite alphabet e.g. from $\mathbb{F}_q$, is generated.
- The *encoder* maps the information vector $\mathbf{u}$ to a codeword $\mathbf{c} = (c_0, \ldots, c_{n-1})$ from a code $\mathcal{C}$, i.e.:

$$\mathbf{u} \in \mathbb{F}_q^k \rightarrow \mathbf{c} \in \mathbb{F}_q^n, \text{ where } c \in \mathcal{C}.$$

- The codeword $\mathbf{c}$ is transmitted over a noisy *channel* where the received vector $\mathbf{r} = \mathbf{c} + \mathbf{e} \in \mathbb{F}_q^n$ where $\mathbf{e}$ is an error vector, is received.
- The *decoder* decodes the received vector, i.e. corrects the errors,

$$\mathbf{r} \in \mathbb{F}_q^n \rightarrow \hat{\mathbf{c}} \in \mathbb{F}_q^n.$$

- Finally, the *sink* maps the codeword back to the information vector,

$$\hat{\mathbf{c}} \in \mathbb{F}_q^n \rightarrow \hat{\mathbf{u}} \in \mathbb{F}_q^k.$$

  If everything was successful, then $\hat{\mathbf{u}} = \mathbf{u}$.

3.3 Linear Block Codes

In this section, linear block codes and their properties are introduced. The term *block* means that each codeword is independent of the previous and the next codewords however, as this is often assumed these codes are simply called linear codes.

Definition 3.5 (Linear Block Code)**.** *[Wac+, p. 41] A linear block code $\mathcal{C}$ with parameters $[n, k, d]_q$ is a k-dimensional linear subspace of the vector space $\mathbb{F}_q^n$ with minimum Hamming distance d. Elements of $\mathcal{C}$ are called codewords.*

As indicated by Definition 3.5, a linear code is characterized by a set of parameters [Wac+, p. 41]:

- n is the length which means the number of symbols in a codeword $\mathbf{c}$, i.e. the number of transmitted symbols.
- k is the dimension which means the number of symbols in an information vector $\mathbf{u}$.
- d is the minimum Hamming distance which means the minimum number of differing symbols in any two codewords. I.e. let $\mathcal{C}$ be a linear code, then the minimum distance d of $\mathcal{C}$ is defined as

  $$d = \min_{\mathbf{a},\mathbf{b}\in\mathcal{C},\ \mathbf{a}\neq\mathbf{b}} \{d_H(\mathbf{a},\mathbf{b})\}.$$

  The minimum distance can be used to compute how many errors a code can correct. In general, a linear code $\mathcal{C}$ can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors.
- q is the number of available symbols i.e. the size of the alphabet.

Additionally, a linear code has the following properties:

- $n - k$ is the redundancy, i.e. the number of elements that do not contain new information but are necessary in order to be able to decode,
- q^k is the number of codewords in the code,
- $\frac{k}{n}$ is the rate of the code. In general, if the rate is big, then few additional redundancy symbols are transmitted which is good for efficient transmission.

Lemma 3.6 (Minimum Distance of a Linear Code)**.** *[Wac+, p. 42] Let $\mathcal{C}$ be a linear code, then the following holds for the minimum distance d of $\mathcal{C}$:*

$$d = \min_{\mathbf{c}\in\mathcal{C},\ \mathbf{c}\neq\mathbf{0}} \{wt(\mathbf{c})\},$$

Proof. The fact that $\mathcal{C}$ is linear means that for two codewords, $\mathbf{a}, \mathbf{b} \in \mathcal{C}$, then $\mathbf{a} - \mathbf{b} = \mathbf{c} \in \mathcal{C}$. If $\mathbf{a}$ and $\mathbf{b}$ are not the same codeword, then $\mathbf{c} \neq \mathbf{0}$. Therefore, it follows that [Wac+, p. 42]

$$d = \min_{\mathbf{a},\mathbf{b}\in\mathcal{C},\ \mathbf{a}\neq\mathbf{b}} \{wt(\mathbf{a}-\mathbf{b})\} = \min_{\mathbf{c}\in\mathcal{C},\ \mathbf{c}\neq\mathbf{0}} \{wt(\mathbf{c})\}.$$

□

A *generator matrix* which is defined in Definition 3.7 plays an important role in the mapping from the information vector $\mathbf{u}$ to a codeword $\mathbf{c}$, i.e. in the *encoder* in the transmission model in Figure 3.1.

Definition 3.7 (Generator Matrix)**.** *[Wac+, p. 46] A $k \times n$ matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ is a generator matrix of a linear $[n, k, d]_q$ code $\mathcal{C}$ if its rows are k linearly independent vectors that form a basis of $\mathcal{C}$.*

An information vector $\mathbf{u} \in \mathbb{F}_q^k$ is mapped to a codeword $\mathbf{c} \in \mathbb{F}_q^n$ from a code $\mathcal{C}$ by computing,

$$\mathbf{c} = \mathbf{uG} \in \mathbb{F}_q^n, \quad c \in \mathcal{C}.$$

Another important matrix is the *parity-check matrix*, but in order to define the parity-check matrix, the *dual code* should first be defined:

Definition 3.8 (Dual Code)**.** *[Wac+, p. 48] Let $\mathcal{C}$ be a linear $[n, k, d]_q$ code, then the dual code of $\mathcal{C}$ is defined as*

$$\mathcal{C}^\perp := \{\mathbf{c}^\perp \in \mathbb{F}_q^n \mid \langle \mathbf{c}, \mathbf{c}^\perp \rangle = 0, \text{ for all } \mathbf{c} \in \mathcal{C}\},$$

where $\langle \mathbf{c}, \mathbf{c}^\perp \rangle = \sum_{i=1}^n c_i c_i^\perp$ is the scalar product.

Definition 3.9 (Parity-Check Matrix)**.** *[Agu+21a, p. 9] Let $\mathcal{C}$ be an $[n, k, d]_q$ code, then an $(n-k) \times n$ matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k) \times n}$ that satisfies both*

$$\mathcal{C} = \{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{Hc}^T = \mathbf{0}\},$$

and

$$\mathcal{C}^\perp = \{\mathbf{uH} \mid \mathbf{u} \in \mathbb{F}_q^{n-k}\},$$

is called a parity-check matrix of $\mathcal{C}$. Note that the latter criteria means that $\mathbf{H}$ is the generator matrix of the dual code $\mathcal{C}^\perp$.

It follows from Definition 3.9 that the following also holds for a generator matrix $\mathbf{G}$ and a parity-check matrix $\mathbf{H}$ [Wac+, p. 50]:

$$\mathbf{G} \cdot \mathbf{H}^T = \mathbf{0}.$$

Using the parity-check matrix, the *syndrome* can be defined:

Definition 3.10 (Syndrome)**.** *[Rot06, p. 34] Let $\mathcal{C}$ be a linear $[n,k,d]_q$ code, and let $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ be a parity-check matrix of $\mathcal{C}$. The syndrome $\mathbf{s}$ of a vector $\mathbf{v} \in \mathbb{F}_q^n$ is defined by*

$$\mathbf{s} = \mathbf{H}\mathbf{v}^T.$$

It follows from Definition 3.9 that

$$\mathbf{v} \in \mathcal{C} \Leftrightarrow \mathbf{H}\mathbf{v}^T = \mathbf{0}.$$

The syndrome plays an important role as the term *decoding a random code* corresponds to solving an instance of the *syndrome decoding problem* which is the following problem: Given a parity-check matrix $\mathbf{H} \in \mathbb{F}_q^{(n-k)\times n}$ and a syndrome $\mathbf{s} \in \mathbb{F}_q^n$, find $\mathbf{e}$ such that [EB21b, p. 1]

$$\mathbf{s} = \mathbf{H}\mathbf{e}^T.$$

Note that the syndrome of a received word $\mathbf{r} = \mathbf{c} + \mathbf{e}$ where $\mathbf{c}$ is the codeword and $\mathbf{e}$ is the error vector as the same as the syndrome of $\mathbf{e}$, since

$$\mathbf{H}\mathbf{r}^T = \mathbf{H}(\mathbf{c}+\mathbf{e})^T = \mathbf{H}\mathbf{c}^T + \mathbf{H}\mathbf{e}^T = \mathbf{0} + \mathbf{H}\mathbf{e}^T = \mathbf{H}\mathbf{e}^T.$$

3.3.1 Linear Cyclic Codes

The set of error-correcting codes that will be covered in this thesis includes cyclic codes. Therefore, (linear) cyclic codes are introduced in this Section.

Definition 3.11 (Cyclic Code)**.** *[MS77, p. 188-189] A code $\mathcal{C}$ is cyclic, if any cyclic shift of a codeword of $\mathcal{C}$ is again a codeword of $\mathcal{C}$, i.e.,*

$$(c_0, c_1, \ldots, c_{n-1}) \in \mathcal{C} \Rightarrow (c_{n-1}, c_0, \ldots, c_{n-2}) \in \mathcal{C}.$$

Definition 3.11 implies that a codeword can be shifted any number of positions and still be a codeword.

Now, associate the vector $(c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$ with the polynomial $c(x) := c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1} \in \mathbb{F}_q[x]$. A shift to the right of this polynomial corresponds to multiplying $c(x)$ with the variable x,

$$c(x) \cdot x = c_0x + c_1x^2 + c_2x^3 + \cdots + c_{n-1}x^n.$$

In order to achieve a cyclic shift and not just a right shift, the rightmost coefficient should become the leftmost coefficient which can be done by computing,

$$\begin{aligned} c(x) \cdot x + c_{n-1} - c_{n-1}x^n &= c(x) \cdot x - c_{n-1} \cdot (x^n - 1) \\ &= c(x) \cdot x \mod (x^n - 1). \end{aligned}$$

This leads to the following fact: A linear code is cyclic if and only if [Wac+, p. 84],

$$c(x) \in \mathcal{C} \Rightarrow c(x) \cdot x \mod (x^n - 1) \in \mathcal{C},$$

and it can be shown that the following also holds for every $a(x) \in \mathbb{F}_q[x]$ [Wac+, p. 84]:

$$c(x) \in \mathcal{C} \Rightarrow c(x) \cdot a(x) \mod (x^n - 1) \in \mathcal{C}. \tag{3.1}$$

A cyclic code can be generated using a *generator polynomial*. Given a generator polynomial, a generator matrix as we know from the previous section, can be constructed.

The following Theorem and Lemma are stated without proofs. Proofs can be found in [Wac+, p. 85].

Theorem 3.12 (Generator Polynomial)**.** *[Wac+, p. 85] Let $\mathcal{C}$ be a cyclic $[n, k, d]_q$ code. Then there is a unique monic polynomial $g(x)$ such that for every $c(x) \in \mathbb{F}_q[x]$ of degree $n - 1$, it holds that*

$$c(x) \in \mathcal{C} \quad \Leftrightarrow \quad g(x) \mid c(x).$$

The polynomial $g(x)$ from the Theorem above is called the generator polynomial and has degree $n-k$. It follows from the Theorem that given a generator polynomial $g(x)$ a cyclic code can be defined by

$$\mathcal{C} = \{u(x) \cdot g(x) \mid u(x) \in \mathbb{F}_q[x] \text{ and } \deg u(x) < k\}. \tag{3.2}$$

Lemma 3.13 (Generator Polynomial)**.** *[Wac+, p. 85] Let $g(x)$ be the generator polynomial of a cyclic $[n, k, d]_q$ code. Then*

$$g(x) \mid (x^n - 1).$$

Furthermore, it also holds the other way around: If $g(x)$ is a polynomial in $\mathbb{F}_q[x]$ for which it holds that $g(x) \mid (x^n - 1)$, then $\mathcal{C}$ from Equation (3.2) is a cyclic code.

Such a generator polynomial $g(x)$ have some roots, and when we talk about the *roots* of a cyclic code $\mathcal{C}$, we mean the roots of its generator polynomial $g(x)$ which is a subset of the roots of $x^n - 1$ as the generator polynomial is a factor of $x^n - 1$ according to

Lemma 3.13. The roots of $x^n - 1$ lie in the splitting field (see Definition 3.2) $\mathbb{F}_{q^m}$, and in no smaller field [MS77, p. 196]. The length n is chosen such that $\gcd(n, q) = 1$, and then the dimension k is derived from n and the number of roots as we say that there are $n - k$ distinct roots. Given a list of roots, say $\alpha_1, \alpha_2, \ldots, \alpha_{n-k}$, the generator polynomial is constructed as

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha_i). \tag{3.3}$$

An element in $\mathbb{F}_{q^m}$ is a root of $g(x)$ if and only if all its *conjugates* with respect to $\mathbb{F}_q$ also are [Rot06, p. 248]. The conjugates of an element in $\mathbb{F}_{q^m}$ are all the elements that belong to the same *conjugacy class* as itself. The term conjugacy class is defined below.

Definition 3.14 (Conjugacy Class)**.** *[Rot06, p. 219] The conjugacy class with respect to $\mathbb{F}_q$ of an element $\alpha \in \mathbb{F}_{q^m}$ is given by,*

$$C_\alpha = \{\alpha, \alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{i-1}}\},$$

where i is the smallest positive integer such that $\alpha^{q^i} = \alpha$.

The different conjugacy classes are either identical or distinct [Wac+, p. 32], and therefore a way to cover all the roots is to take the union of all the conjugacy classes:

$$\{\alpha_1, \alpha_2, \ldots, \alpha_{n-k}\} = C_{\alpha_1} \cup C_{\alpha_2} \cup \cdots \cup C_{\alpha_{n-k}},$$

A way to construct a generator polynomial as in Equation (3.3) from the union of conjugacy classes is in the following way:

$$g(x) = \prod_{i=1}^{n-k} M_{\alpha_i}(x),$$

where $M_{\alpha_i}(x)$ is the *minimal polynomial* with respect to $\mathbb{F}_q$ of each element in C_{α_i} as defined in Definition 3.15.

Definition 3.15 (Minimal Polynomial)**.** *[Rot06, p. 219] Let $\alpha \in \mathbb{F}_{q^m}$ and let C_α be the conjugacy class that contains α as defined in Definition 3.14. The minimal polynomial of α with respect to $\mathbb{F}_q$ is defined by,*

$$M_\alpha(x) = \prod_{\gamma \in C_\alpha} (x - \gamma).$$

To summarize: Given a set of roots, a generator polynomial can be constructed. Given a generator polynomial, a cyclic code can be generated.

Finally, writing $g(x) = g_0 + g_1 x + \cdots + g_{n-k}x^{n-k}$, it follows that the corresponding generator matrix can be constructed as

$$\mathbf{G}_{BCH} = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & 0 & 0 & 0 \\ 0 & g_0 & g_1 & \cdots & g_{n-k} & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & g_0 & g_1 & \cdots & g_{n-k} \end{pmatrix} \in \mathbb{F}_q^{k \times n}.$$

3.4 Reed-Solomon Codes

Reed-Solomon (RS) codes are a class of codes that is used a lot in practice since it has some great advantages. First of all, RS codes meet the Singleton bound (see Theorem 3.16) which means that they have the best, i.e. largest, possible minimum distance for their length n and dimension k, $d = n - k + 1$.

Theorem 3.16 (Singleton Bound)**.** *[RB20, Chapter. 3, p. 17] For any $[n, k, d]_q$ linear code, then it holds that*

$$d \leq n - k + 1.$$

Furthermore, RS codes can be efficiently encoded and decoded for up to $\lfloor \frac{d-1}{2} \rfloor$ errors. However, there is also the downside that their length n is bounded from above by the field size, $n \leq q - 1$.

3.4.1 Generalized Reed-Solomon Codes

Definition 3.17 (Generalized Reed-Solomon (GRS) Code)**.** *[Rot06, p. 148] Let $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_q$ be n distinct non-zero elements called code locators, and let $v_1, v_2, \ldots, v_n \in \mathbb{F}_q$ be non-zero elements called column multipliers (not necessarily distinct), where $n \leq q - 1$. A Generalized Reed-Solomon code of length n and dimension k is defined by the following $(n - k) \times n$ parity-check matrix,*

$$\mathbf{H}_{GRS} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{n-k-1} & \alpha_2^{n-k-1} & \cdots & \alpha_n^{n-k-1} \end{pmatrix} \begin{pmatrix} v_1 & 0 & \cdots & 0 \\ 0 & v_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_n \end{pmatrix},$$

and is denoted $\mathcal{RS}(n, k)$.

It follows that GRS codes have the following generator matrix. For a proof, see [Rot06, p. 149]

Definition 3.18 (Generator Matrix of GRS codes)**.** *[Rot06, p. 149] Given an $\mathcal{RS}(n,k)$ code defined by its parity-check matrix $\mathbf{H}_{GRS}$ as in Definition 3.17. There exist non-zero elements $v'_1, v'_2, \ldots, v'_n$ such that*

$$\mathbf{G}_{GRS} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \cdots & \alpha_n^{k-1} \end{pmatrix} \begin{pmatrix} v'_1 & 0 & \cdots & 0 \\ 0 & v'_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v'_n \end{pmatrix},$$

is a generator matrix with $\mathbf{G}_{GRS}\mathbf{H}_{GRS}^T = \mathbf{0}$.

The GRS codes can also be defined by evaluating degree-restricted polynomials as:

Theorem 3.19 (GRS Codes defined as Polynomial Evaluation)**.** *[Rot06, p. 150] Let $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_q$ be n distinct non-zero elements called code locators, and let $v_1, v_n, \ldots, v_n \in \mathbb{F}_q$ be non-zero elements called column multipliers (not necessarily distinct), where $n \leq q-1$. Then, the code which is generated by the following set of vectors,*

$$\{eval(u(x)) := (v'_1 u(\alpha_0), v'_2 u(\alpha_1), \ldots, v'_n u(\alpha_{n-1})) \mid u(x) \in \mathbb{F}_q[x] \text{ and } \deg u(x) < k\},$$

is an $\mathcal{RS}(n,k)$ Generalized Reed-Solomon code.

Let $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ be an information vector that is encoded to the codeword $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ using the usual calculation:

$$\mathbf{c} = \mathbf{uG} \in \mathbb{F}_q^n, \quad c \in \mathcal{C}. \tag{3.4}$$

This calculation is related to Theorem 3.19 in the following way: Interpret the information vector $\mathbf{u}$ as the polynomial,

$$u(x) = u_0 + u_1 x + \cdots + u_{k-1} x^{k-1} \in \mathbb{F}_q[x]_{<k},$$

and then the calculation in Equation (3.4) corresponds exactly to the evaluation in Definition 3.19:

$$\mathbf{uG} = (v'_1 u(\alpha_0), v'_2 u(\alpha_1), \ldots, v'_n u(\alpha_{n-1})).$$

3.4.2 Conventional Reed-Solomon Codes

The class of codes called *BCH* codes which will later be introduced is related to *conventional* RS codes which will be introduced in this Section. Conventional RS codes are a special case of GRS codes obtained as:

Definition 3.20 (Conventional Reed-Solomon Codes)**.** *[Rot06, p. 151] Let n be a positive integer dividing $q-1$ and let α be an element of multiplicative order n in $\mathbb{F}_q$. Furthermore, let b be an integer. A Conventional Reed-Solomon code over $\mathbb{F}_q$ is a GRS code with code locators,*

$$\alpha_j = \alpha^{j-1}, \qquad 0 \leq j \leq n-1,$$

and column multipliers,

$$v_j = v^{b(j-1)}, \qquad 0 \leq j \leq n-1.$$

Lemma 3.21 (Parity-Check Matrix of Conventional RS Codes)**.** *[Rot06, p. 151] A conventional RS code $\mathcal{RS}(n,k)$ has the following parity-check matrix,*

$$\mathbf{H}_{RS} = \begin{pmatrix} 1 & \alpha^{b} & \cdots & \alpha^{(n-1)b} \\ 1 & \alpha^{b+1} & \cdots & \alpha^{(n-1)(b+1)} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{b+d-2} & \cdots & \alpha^{(n-1)(b+d-2)} \end{pmatrix},$$

where $d-1 = n-k$.

Proof. The Lemma follows by plugging in the code locators and column multipliers from Definition 3.20 in the parity-check matrix for GRS codes in Definition 3.17. □

As usual, associate the vector $(c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}_q^n$ with the polynomial $c(x) := c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1}x^{n-1} \in \mathbb{F}_q[x]$. As explained in Section 3.3, for every codeword $\mathbf{c}$, it holds that

$$\mathbf{c} \in \mathcal{C}_{RS} \quad \Leftrightarrow \quad \mathbf{c}\mathbf{H}_{RS}^T = \mathbf{0}.$$

From the calculation,

$$\mathbf{c}\mathbf{H}_{RS}^T = \begin{pmatrix} c_0 + c_1\alpha^{b} + \cdots + c_{n-1}\alpha^{(n-1)b} \\ c_0 + c_1\alpha^{b+1} + \cdots + c_{n-1}\alpha^{(n-1)(b+1)} \\ \vdots \\ c_0 + c_1\alpha^{b+d-2} + \cdots + c_{n-1}\alpha^{(n-1)(b+d-2)}, \end{pmatrix}^T,$$

it follows that $\mathbf{cH}_{RS}^T = \mathbf{0}$ can only be achieved if $c(\alpha^\ell) = 0$ for $\ell = b, b+1, \ldots, b+d-2$. Hence,

$$\mathbf{c} \in \mathcal{C}_{RS} \quad \Leftrightarrow \quad c(\alpha^\ell) = 0, \text{ for } \ell = b, b+1, \ldots, b+d-2. \tag{3.5}$$

As mentioned, conventional RS codes are related to BCH codes, both of which are cyclic codes. Therefore, conventional RS codes can be generated using a generator polynomial as described in Section 3.3.1: The generator polynomial $g(x)$ of $\mathcal{C}_{RS}$ is defined by plugging in the roots in Equation (3.5). Hence,

$$g(x) = (x - \alpha^b)(x - \alpha^{b+1}) \cdots (x - \alpha^{b+d-2}).$$

It follows from Equation (3.5)

$$\mathbf{c} \in \mathcal{C}_{RS} \quad \Leftrightarrow \quad g(x) \mid c(x).$$

This means that a conventional RS code can be generated by its generator polynomial exactly as in Section 3.3.1.

3.4.3 Decoding Reed-Solomon Codes

There exists different decoders for Reed-Solomon codes, but in this project a decoder called *Interpolation-Based Unique Decoding* is used. As the name says, the decoder is interpolation-based, and it consists of the following two steps:

1. Interpolation of a bivariate polynomial: The goal of this step is to find a bivariate polynomial $Q(x, y) = Q_0(x) + Q_1(x) \cdot y$ satisfying some conditions (see Theorem 3.22).
2. Factorization of the bivariate polynomial: The goal of this step is to use the bivariate polynomial from the interpolation step to find the information vector/polynomial $u(x)$ (see Theorem 3.23).

Theorem 3.22 (Interpolation Step)**.** *[RB20, Chapter 5, p. 24] Let $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ be the code locators of a $\mathcal{GRS}(n, k)$ code, and let $\tau = \lfloor \frac{n-k}{2} \rfloor$. Furthermore, let $\mathbf{r} = (r_1, r_2, \ldots, r_n) = \mathbf{c} + \mathbf{e}$ be a received word with $\mathbf{c} = eval(u(x))$ and $wt(\mathbf{e}) \leq \tau$. Then there exists a non-zero polynomial $Q(x, y) = Q_0(x) + y \cdot Q_1(x) \in \mathbb{F}_q[x, y]$ such that*

1. *$Q(\alpha_i, r_i) = 0$ for $i = 1, \ldots, n$,*
2. *$\deg Q_0 \leq \tau + k - 1$,*
3. *$\deg Q_1 \leq \tau$.*

Proof. Condition 1 simply means that it is a linear system with n equations. The number of unknowns in this linear system of equations is the number of coefficients of the polynomials $Q_0(x)$ and $Q_1(x)$. Since $\deg Q_0 \leq \tau + k - 1$, then $Q_0(x)$ has $\tau + k$ coefficients, and since $\deg Q_1 \leq \tau$, then $Q_1(x)$ has $\tau + 1$ coefficients. Hence, the total number of coefficients, i.e. the total number of unknowns is,

$$\begin{aligned}(\tau + k) + (\tau + 1) &= 2\tau + k + 1 \\ &= 2 \cdot \lfloor \frac{n-k}{2} \rfloor + k + 1 \\ &\geq n - k - 1 + k + 1 \\ &= n.\end{aligned}$$

From second to third step above, the following is used:

$$2 \cdot \lfloor \frac{n-k}{2} \rfloor = \begin{cases} n-k & \text{if} \quad n-k \equiv 0 \mod 2 \\ n-k-2 \cdot \frac{1}{2} = n-k-1 & \text{if} \quad n-k \equiv 1 \mod 2 \end{cases}$$

Hence, since the number of unknowns is at least as big as the number of equations, there exists always such a polynomial $Q(x, y)$. □

Theorem 3.23 (Factorization Step)**.** *[RB20, Chapter 5, p. 24] For any such $Q(x, y)$ as defined in Theorem 3.22, it holds that $u(x) = -Q_0(x)/Q_1(x)$.*

Proof. First of all, the bivariate polynomial $Q(x, y)$ satisfies $Q(\alpha_i, u(\alpha_i) + e_i) = 0$ which is condition 1 of Theorem 3.22. Since $wt(\mathbf{e}) \leq \tau$, then $e_i = 0$ for at least $n - \tau$ positions, and hence the univariate polynomial $Q(x, u(x)) = Q_0(x) + Q_1(x) \cdot u(x)$ has at least $n - \tau$ roots, α_i where $u(\alpha_i) = r_i$. Since $Q(x, u(x))$ has at least $n - \tau$ roots, then $Q(x, u(x))$ must also have degree at least $n - \tau$. On the other hand,

$$\begin{aligned}\deg Q(x, u(x)) &\leq \max\{\deg Q_0, \deg Q_1 + \deg u(x)\} \\ &\leq \max\{\tau + k - 1, \tau + k - 1\} \\ &= \tau + k - 1 \\ &< n - \tau,\end{aligned}$$

which is a contradiction, so in order to fulfill both constraints, it must be the case that $Q(x, u(x)) = 0$. The conclusion is that $Q_0(x) + u(x)Q_1(x) = 0$ and $u(x) = -Q_0(x)/Q_1(x)$. □

The algorithm is summarized in Algorithm 1 where α_i and r_i for $i = 1, \ldots, n$ have been plugged in. For a proof of the correctness of the algorithm, see [RB20, p. 25].

Algorithm 1 Interpolation-Based Unique Decoding [Wac+, p. 78]

1: **Input:** received word $\mathbf{r} = (r_1, r_2, \ldots, r_n)$
2: **Output:** message word $\mathbf{u}$
3: **Interpolation step:** solve the following linear system of equations:

$$\begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \cdots & \alpha_1^{\tau+k-1} & r_1 & r_1 \cdot \alpha_1 & \cdots & r_1 \cdot \alpha_1^{\tau} \\ 1 & \alpha_2 & \alpha_2^2 & \cdots & \alpha_2^{\tau+k-1} & r_2 & r_2 \cdot \alpha_2 & \cdots & r_2 \cdot \alpha_2^{\tau} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \cdots & \alpha_n^{\tau+k-1} & r_n & r_n \cdot \alpha_n & \cdots & r_n \cdot \alpha_n^{\tau} \end{pmatrix} \cdot \begin{pmatrix} Q_{0,0} \\ Q_{0,1} \\ \vdots \\ Q_{0,\tau+k-1} \\ Q_{1,0} \\ Q_{1,1} \\ \vdots \\ Q_{1,\tau} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}.$$

4: set $Q_0(x) = \sum_{i=0}^{\tau+k-1} Q_{0,i} x^i$ and $Q_1(x) = \sum_{i=0}^{\tau} Q_{1,i} x^i$
5: **Factorization step:** calculate $u(x) = \frac{-Q_0(x)}{Q_1(x)}$
6: **return** message word $\mathbf{u}$

3.5 BCH Codes

The next class of codes that will be introduced was already mentioned in the context of conventional RS codes and is called *BCH codes*. Recall that both conventional RS codes and BCH codes are linear cyclic codes as described in Section 3.3.1, and BCH codes are related to conventional RS codes in the sense that they are *subfield subcodes* of RS codes. What this means will be explained later.

Let n be a positive integer chosen such that $\gcd(n, q) = 1$. Let m be the smallest possible positive integer such that $n \mid q^m - 1$. Furthermore, let $\alpha \in \mathbb{F}_{q^m}$ be an element of multiplicative order n, and let b and δ be integers with $0 < \delta \leq n$. The BCH code $\mathcal{C}_{BCH}$ over $\mathbb{F}_q$ is then defined as a linear cyclic $[n, k]_q$ (the minimum distance will be derived later) code whose set of roots consists of the elements of the consecutive sequence [Rot06, p. 162],

$$\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+\delta-2}, \tag{3.6}$$

and their conjugates. In other words, $\mathcal{C}_{BCH}$ consists of all the polynomials $c(x) \in$

$\mathbb{F}_q[x]$ of degree at most n where [Rot06, p. 162],

$$c(\alpha^i) = 0, \qquad \text{for } i = b, b+1, \ldots, b+\delta-2. \tag{3.7}$$

This root sequence looks similar to the one for the conventional RS code. However, the difference between the BCH code and the conventional RS code is that the BCH code is over $\mathbb{F}_q$ while the conventional RS code is over the extension field $\mathbb{F}_{q^m}$. Therefore, the set of roots of the BCH code consists of some conjugates that are not included in the set of roots of the conventional RS code. Including the extra conjugates is a way of converting from the extension field $\mathbb{F}_{q^m}$ to the base field $\mathbb{F}_q$.

Recall that the generator polynomial of the conventional RS code is defined as,

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

The conjugates are computed and included in the generator polynomial as described in Section 3.3.1: Define a list of conjugacy classes such that

$$\{\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+\delta-2}\} = C_{\alpha^b} \cup C_{\alpha^{b+1}} \cup \cdots \cup C_{\alpha^{b+\delta-2}},$$

where C_{α^i} is a conjugacy class that contains the root α^i of $\mathcal{C}$. Then, define the generator polynomial as

$$g(x) = \prod_{i=b}^{b+\delta-2} M_{\alpha^i}(x),$$

where $M_{\alpha^i}(x)$ is the minimal polynomial with respect to $\mathbb{F}_q$ of each element in C_{α^i} as defined in Definition 3.15. The dimension of the BCH code can be derived from:

$$\deg g(x) = n - k. \tag{3.8}$$

What remains is to derive the minimum distance of the BCH code: Even though it is easy to determine the minimum distance of an RS code from the Singleton bound in Theorem 3.16, it is not as easy to determine the minimum distance of a BCH code. However, the following Theorem gives a lower bound on the minimum distance:

Theorem 3.24 (The BCH Bound)**.** *[Wac+, p. 91] Let $\mathcal{C}_{BCH}$ be an $[n, k, d]_q$ BCH code with generator polynomial $g(x)$ where n divides $q^m - 1$ and $\alpha \in \mathbb{F}_{q^m}$ is an element of order n. Let $b, b+1, \ldots, b+\delta-2$ and their conjugates be the underlying root sequence for some integers b and $\delta \geq 2$. Then $d \geq \delta$.*

Proof. The polynomial,

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

clearly divides $g(x)$ since $b, b+1, \ldots, b+\delta-2$ is a subset of all the roots of the BCH code. Furthermore, the polynomial $g'(x)$ is the generator polynomial of a conventional RS code with parameters $[n, n-\delta+1, \delta]_{q^m}$ and code locators and column multipliers as defined in Definition 3.20. Therefore, it follows that every codeword of $\mathcal{C}_{BCH}$ is also a codeword of the $[n, n-\delta+1, \delta]_{q^m}$ RS code, and hence $\mathcal{C}_{BCH}$ is a *subcode* of the $[n, n-\delta+1, \delta]_{q^m}$ RS code. This also means that the minimum distance of $\mathcal{C}_{BCH}$ is at least the one of the RS code since the codewords in $\mathcal{C}_{BCH}$ with smallest distance are also contained in the RS code. Hence, it is proved that $d \geq \delta$. □

Finally, it will be shown that the BCH code is a subfield subcode of a conventional RS code as earlier claimed:

Theorem 3.25 (BCH Codes as Subfield Subcodes)**.** *[Wac+, p. 91] A BCH code with parameters $[n, k, d \geq \delta]_q$ and generator polynomial $g(x)$ is a subfield subcode of a conventional RS code with parameters $[n, n-\delta+1, \delta]_{q^m}$.*

Proof. In the proof of Theorem 3.24, it was shown that every codeword of $\mathcal{C}_{BCH}$ is also a codeword of $\mathcal{RS}[n, n-\delta+1, \delta]_{q^m}$, i.e.,

$$\mathcal{C}_{BCH} \subseteq (\mathbb{F}_q^n \cap \mathcal{RS}[n, n-\delta+1, \delta]_{q^m}).$$

It can be shown that also "$\supseteq$" holds, i.e. that any RS codeword with coefficients in the base field $\mathbb{F}_q$ is also a codeword in $\mathcal{C}_{BCH}$. This is not proved here, but a proof can be found in e.g. [Wac+, p. 91].

Since both "$\subseteq$" and "$\supseteq$" hold, it follows that

$$\mathcal{C}_{BCH} = (\mathbb{F}_q^n \cap \mathcal{RS}[n, n-\delta+1, \delta]_{q^m}).$$

The conclusion is that all codewords of the RS code that lie in the subfield $\mathbb{F}_q$ are also codewords of the BCH code which proves that BCH codes are subfield subcodes of RS codes. □

Since BCH codes are subfield subcodes of RS codes, then the BCH codes can be decoded using the decoder for RS codes that was presented in Section 3.4.3.

3.6 Binary Repetition Codes

A binary repetition code is probably the most simple error-correcting code that exists. As the name reveals, it simply repeats the information symbols several times.

Definition 3.26 (Binary Repetition Code)**.** *[Wac+, p. 43] A binary repetition code encodes an information vector of length 1,* $\mathbf{u} = (u_0) \in \mathbb{F}_2$*, into a codeword by repeating it* n *times,* $\mathbf{c} = (u_0, \ldots, u_0) \in \mathbb{F}_2^n$*. In other words, the binary repetition code is constructed by the following generator matrix,*

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \in \mathbb{F}_2^{1 \times n}.$$

It follows by construction that a binary repetition code has length n, dimension $k = 1$, and that it consists of only the two codewords,

$$\mathcal{C} = \{(00 \ldots 0), (11 \ldots 1)\}.$$

This also means that the minimum distance is n. In total, a binary repetition code has parameters $[n, 1, n]_2$.

A binary repetition code decodes and corrects errors using *majority decision*: If a codeword contains more 1's than 0's, then it is decoded to a 1, and if it contains more 0's than 1's, then it is decoded to a 0. It follows that a binary repetition code can correct $\lfloor \frac{n-1}{2} \rfloor$ errors.

3.7 Reed-Muller Codes

Reed-Muller codes are an old and well understood class of codes. They can be seen as a generalization of Reed-Solomon codes. Recall that for RS codes, a list of n code locators, $\alpha_1, \ldots, \alpha_n$ were used. For an RM code, one code locator is extended to a vector of length m, so instead of $\alpha_i \in \mathbb{F}_q$, RM codes use $\alpha_i = (\alpha_{i_1}, \ldots, \alpha_{i_m}) \in \mathbb{F}_q^m$ [RB20, Chapter. 8, p. 64]. We say that RM codes operate with m variables. Another difference between RS codes and RM codes is that RS codes operate over any field $\mathbb{F}_q$ whereas it is most common to only work with binary RM codes, i.e. RM codes over the binary field $\mathbb{F}_2$. In this thesis, only binary RM codes will be considered. There are two parameters that define an RM code, the number of variables m which was already introduced as the length of a code locator, and a parameter r that determines the order or degree of the RM code. Let $v_1, \ldots, v_m$ denote the m variables. Since we work with binary RM codes, each variable can be assigned either a 1 or a 0. The parameter r determines the allowed degree of each term of the polynomial. For a first-order RM code, i.e. an RM code with $r = 1$, only terms of degree at most 1 are considered. Hence, for e.g. $m = 4$ (note that the notation below means 5 bits),

$$\mathbf{1}\ v_4\ v_3\ v_2\ v_1 \in \mathbb{F}_2^5.$$

Note that the **1** denotes the 0-order term. For a second-order RM code, i.e. an RM code with $r = 2$, all terms of degree at most 2 are considered, hence for e.g. $m = 4$,

$$\mathbf{1}\ v_4\ v_3\ v_2\ v_1\ v_3v_4\ v_2v_4\ v_1v_4\ v_2v_3\ v_1v_3\ v_1v_2 \in \mathbb{F}_2^{11}.$$

The parameters, m and r, together define an $\mathcal{RM}(r, m)$ code.

Since RM codes operate over the binary field $\mathbb{F}_2$, a straightforward way to define them is in terms of binary vectors of length m (corresponding to assignments of the m variables to either 0's or 1's) as in the following Definition 3.27.

Definition 3.27 (First-Order Reed-Muller Code)**.** *[Wac+, p. 95] A binary first-order $\mathcal{RM}(1, m)$ is defined by a generator matrix which contains all 2^m binary vectors of length m as columns and additionally the all-one row.*

For $m = 4$, such a generator matrix becomes,

$$\mathbf{G}_{RM(1,4)} = \begin{array}{c} \\ \mathbf{1} \\ \mathbf{v_4} \\ \mathbf{v_3} \\ \mathbf{v_2} \\ \mathbf{v_1} \end{array} \left[\begin{array}{cccccccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right] \in \mathbb{F}_2^{5\times 16}. \tag{3.9}$$

Note that the line between the first two rows has simply been added in order to show the distinction between the all-one row and the 2^m column-vectors of length m as described in Definition 3.27. The order of the terms is marked to the left of the matrix, i.e. the 0-order term comes first, then the v_4 term followed by v_3, etc.

An example of encoding is given: Let $v_1 = v_2 = v_4 = 1$ and $v_3 = 0$, then the information bit string becomes (the first term, i.e. the 0-order term, can freely be chosen as either 0 or 1. It is chosen as 1 in this case):

$$\mathbf{u} = \mathbf{1}v_4v_3v_2v_1 = 11011 \in \mathbb{F}_2^5.$$

Multiplying with the generator matrix $\mathbf{G}_{RM(1,4)}$, the codeword becomes:

$$\mathbf{c} = \mathbf{u} \cdot \mathbf{G}_{RM(1,4)} = 1001100101100110 \in \mathbb{F}_2^{16}.$$

The example above indicates that an $\mathcal{RM}(1, 4)$ code has dimension $k = 5$ (the number of rows of $\mathbf{G}_{RM(1,4)}$ and the length of $\mathbf{u}$) and length $n = 16$ (the number of columns of $\mathbf{G}_{RM(1,4)}$ and the length of $\mathbf{c}$). A more formal introduction of the code parameters for RM codes will later be given. In order to prove the minimum distance of RM codes later, it is necessary to include another Theorem of first-order RM codes which describes a recursive procedure. The following Theorem is stated without proof as its extension Theorem to higher order RM codes is shown and proved later.

Definition 3.28 (Recursive Construction of First-Order Reed-Muller Code)**.** *[Wac+, p. 96] Given an $\mathcal{RM}(1,m)$ code, we can construct an $\mathcal{RM}(1,m+1)$ code as follows:*

$$\mathcal{RM}(1,m+1) = \{(\mathbf{u}, \mathbf{u}+\mathbf{v}) : \ \mathbf{u} \in \mathcal{RM}(1,m), \ \mathbf{v} \in \mathcal{C}_{RP}\},$$

where $\mathcal{C}_{RP} = \{(0,\ldots,0),(1,\ldots,1)\} \in \mathbb{F}_2^{2^m}$ is a repetition code of length 2^m.

When extending the generator matrix $\mathbf{G}_{RM(1,4)}$ in Equation (3.9) to a generator matrix for a second-order RM code, $\mathbf{G}_{RM(2,4)}$, then the $\binom{m}{r} = \binom{4}{2} = 6$ second-order terms are added, i.e. the terms v_3v_4, v_2v_4, v_1v_4, v_2v_3, v_1v_3, and v_1v_2:

$$\mathbf{G}_{RM(2,4)} = \begin{array}{c|cccccccccccccccc} \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathbf{v_4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \mathbf{v_3} & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \mathbf{v_2} & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ \mathbf{v_1} & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \hline \mathbf{v_3v_4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \mathbf{v_2v_4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ \mathbf{v_1v_4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ \mathbf{v_2v_3} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ \mathbf{v_1v_3} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ \mathbf{v_1v_2} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{array} \in \mathbb{F}_2^{11\times 16}.$$

This generator matrix indicates that an $\mathcal{RM}(2,4)$ code has dimension $k = 11$ (the number of rows) and length $n = 16$ (the number of columns). The following Theorem is an extension of Definition 3.28 to higher order RM codes, and it also presents the code parameters for RM codes.

Theorem 3.29 (Recursive Construction of Reed-Muller Codes)**.** *[MS77, p. 374] Given an $\mathcal{RM}(r+1,m)$ code and an $\mathcal{RM}(r,m)$ code, we can construct an $\mathcal{RM}(r+1,m+1)$ code of order $r+1$ as follows:*

$$\mathcal{RM}(r+1,m+1) = \{(\mathbf{u}, \mathbf{u}+\mathbf{v}) : \ \mathbf{u} \in \mathcal{RM}(r+1,m), \ \mathbf{v} \in \mathcal{RM}(r,m)\}.$$

This gives a $[2^m, \sum_{i=0}^{r}\binom{m}{i}, 2^{m-r}]_2$ code.

Proof. The proof follows from Theorem 3.30. □

The following Theorem 3.30 is stated without a proof. A proof can be found in e.g. [MS77, p. 76]

Theorem 3.30 (Recursive Construction, $(\mathbf{u}, \mathbf{u}+\mathbf{v})$-Construction)**.** *[MS77, p. 76] Given an $[n,k_u,d_u]_2$ code $\mathcal{C}_u$ and an $[n,k_v,d_v]_2$ code $\mathcal{C}_v$, then*

$$\mathcal{C} := \{(\mathbf{u}, \mathbf{u}+\mathbf{v}) : \ \mathbf{u} \in \mathcal{C}_u, \ \mathbf{v} \in \mathcal{C}_v\}$$

is a $[2n, ku+kv, \min\{2d_u, d_v\}]_2$ code.

The parameters of $\mathcal{RM}(r, m)$ codes are summarized below:

- the length is $n = 2^m$,
- the dimension is $k = \sum_{i=0}^{r} \binom{m}{i}$ which can also be explained by the following: It follows from Definition 3.27 that the dimension of first-order RM codes is $m + 1 = \sum_{i=0}^{1} \binom{m}{i}$. We have seen that when extending from order $r - 1$ to order r, then $\binom{m}{r}$ rows are added to the generator matrix which means that the dimension is also increased by $\binom{m}{r}$. Therefore, it follows that $k = \sum_{i=0}^{r} \binom{m}{i} = 1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r}$.
- the minimum distance is $d = 2^{m-r}$.

The decoding algorithm that will be used in this project is called the *Reed decoding algorithm*. The Reed decoding algorithm can be seen as an extension of the majority decision decoding that was used for repetition codes. One bit of the information vector $\mathbf{u}$ is decoded at a time, and each element is decoded by computing different votes for the value and then choose the value that the majority of the votes chose. In this context, one vote is constructed as the sum of a subset of the bits of the codeword $\mathbf{c} = c_0 c_1 \cdots c_{n-1}$.

A description of the general construction of such votes is not easily given. Below is given an example of how to construct a single vote for a 1-order bit of $\mathbf{u}$. For a more detailed description of the algorithm, see e.g. [MS77, p. 387].

Consider an $\mathcal{RM}(1, 4)$ code. Recall that the generator matrix is

$$\mathbf{G}_{RM(1,4)} = \begin{array}{c} \mathbf{1} \\ \mathbf{v_4} \\ \mathbf{v_3} \\ \mathbf{v_2} \\ \mathbf{v_1} \end{array} \left[\begin{array}{cccccccccccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right] \in \mathbb{F}_2^{5\times 16}.$$

Let the information bit string be given as

$$\mathbf{u} = u_0 u_4 u_3 u_2 u_1,$$

and the corresponding codeword becomes

$$\mathbf{c} = \mathbf{u} \cdot \mathbf{G}_{RM(1,4)} = c_0 c_1 \cdots c_{n-1}.$$

Note that as usual the indices of the bits in $\mathbf{u}$ are ordered such that they match the order of the rows in $\mathbf{G}_{RM(1,4)}$ while the bits in $\mathbf{c}$ are simply ordered from 0 to $n - 1$.

One bit of $\mathbf{u}$ is decoded at a time from back to front. Hence, the first step is to decode u_1 by finding some votes. Consider the two first columns of $\mathbf{G}_{RM(1,4)}$ and imagine that only this part of the generator matrix was multiplied with $\mathbf{u}$:

$$u_0u_4u_3u_2u_1 \cdot \left[\begin{array}{cc} 1 & 1 \\ \hline 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{array}\right] = \begin{bmatrix} u_0 & u_0 + u_1 \end{bmatrix} = \begin{bmatrix} c_0 & c_1 \end{bmatrix}$$

$$\Rightarrow c_0 + c_1 = 2u_0 + u_1 = \begin{cases} 1 & \text{if } u_1 = 1 \\ 0 & \text{if } u_1 = 0 \end{cases}$$

The calculation above shows that $c_0 + c_1$ is equal to 1 if and only if $u_1 = 1$. In other words, $c_0 + c_1$ is one vote for the value of u_1. Given the addition of an error vector, there is a risk of voting for a wrong value. This is exactly why more than one vote is computed, and the final decoded bit is decided by the majority of the votes. For each of the first-order bits $u_1, \ldots, u_4$, a total of $2^{m-r} = 2^{4-1} = 8$ votes are constructed, and hence even if 3 errors occurred, the majority of the 8 votes would still vote for the correct value. Hence, it makes sense that an $\mathcal{RM}(1,4)$ code can correct at most $\lfloor \frac{d-1}{2} \rfloor = \lfloor \frac{2^{4-1}-1}{2} \rfloor = \lfloor \frac{7}{2} \rfloor = 3$ errors.

Not all bits of $\mathbf{u}$ can be decoded just like that. When the r-order bits have been decoded, then some changes need to be made: The problem should be reduced to an RM instance of order $r - 1$. This means that when $u_1, \ldots, u_4$ have been decoded, then the problem is reduced to an instance of 0-order RM code by subtracting the already decoded part from the current problem. This is done by computing

$$\begin{aligned} \mathbf{c}' &= \mathbf{c} - u_1v_1 - u_2v_2 - u_3v_3 - u_4v_4 \\ &= u_0\mathbf{1} + \text{error}. \end{aligned}$$

The bit u_0 is now determined by a single vote: If there are more 1's than 0's in $\mathbf{c}'$, then $u_0 = 1$, and else $u_0 = 0$ [MS77, p. 386-287].

The decoding algorithm is summarized below.

Algorithm 2 Reed Decoding Algorithm [MS77, p. 386-387]

1: **Input:** A codeword $\mathbf{c} = c_0 c_1 \dots c_{n-1}$ from an $\mathcal{RM}(r, m)$ code with generator matrix $\mathbf{G}$
2: **Output:** Information vector $\mathbf{u}$
3: $\mathbf{u} \leftarrow$ ""
4: **while** $r \geq 0$: **do** ▷ decode the $\binom{m}{r}$ r-order bits of $\mathbf{u}$
5: $\mathbf{u}_{\text{part}} \leftarrow$ ""
6: For each of the $\binom{m}{r}$ r-order bits of $\mathbf{u}$, construct at set `votes`. Prepend (i.e. insert in front) the majority bit of `votes` to $\mathbf{u}_{\text{part}}$.
7: When all r-order bits are decoded, subtract the already decoded part from the codeword
8: delete the last $\binom{m}{r}$ rows of $\mathbf{G}$
9: $r \leftarrow r - 1$
10: $\mathbf{u}.prepend(\mathbf{u}_{part})$
11: **end while**
12: **return u**

Except for first-order RM codes and RM codes with short length, RM codes have a lower minimum distance than BCH codes. However, an advantage of RM codes is their efficient decoding.[MS77, p. 370]

3.8 Shortening

Shortening is a process of obtaining a shorter code from an existing code. Given an $[n, k, d]_q$ code $\mathcal{C}$, the following procedure shortens the code by one position [Wac+, p.57]:

1. Remove all codewords of $\mathcal{C}$ that does not have a "0" at the first position.
2. Remove this "0" from all the remaining codewords.

The shortened code gets the following parameters:

Lemma 3.31 (Parameters of Shortened Code)**.** *[Wac+, p. 57] Shortening a non-degenerate $[n, k, d]_q$ code $\mathcal{C}$ (non-degenerate means that not all codewords have "0" at the first position) as described above gives an $[n-1, k-1, \geq d]_q$ code $\mathcal{C}_s$.*

Proof. Since 1 position is removed from each codeword, the length of $\mathcal{C}_s$ clearly becomes $n - 1$. In order to prove the dimension, the following fact is used: In a q-ary linear code, for any fixed position j it holds that $\frac{1}{q}$ of the codewords of $\mathcal{C}$ have a "0"

at position j. Therefore, it follows that $|\mathcal{C}_s| = \frac{1}{q}|\mathcal{C}|$. Since the number of codewords in a linear code in general is expressed as the field size to the power of the dimension, as stated in Subsection 3.3, then

$$|\mathcal{C}_s| = \frac{1}{q} \cdot q^k = q^{k-1},$$

and hence the dimension of $\mathcal{C}_s$ is $k-1$. The minimum distance of $\mathcal{C}_s$ is at least as big as the minimum distance of $\mathcal{C}$, because codewords are removed, and no new codewords are added. Hence, if the minimum distance changes, then it can only increase, not decrease. □

Step 1 of the shortening procedure, i.e. fixing the first position to "0", corresponds to removing the first row of the generator matrix $\mathbf{G}$, step 2 of the shortening procedure, i.e. removing the fixed first position, corresponds to removing the first column of the $\mathbf{G}$ [Wac+, p. 57]. Note that a code can be shortened by any number of positions, the procedure above is just repeated the preferred amount of times. When shortening a code from a specific class, e.g. from the RS code class, then sometimes the shortened code belongs to the same class. However, it can also happen that the shortened code does not belong to the same class. An example is the following: Recall that a criteria for a BCH code is that $n \mid q^m - 1$. It is very likely that a shortened BCH code does not satisfy this criteria which means that it is possible that a shortened BCH code is not a BCH code anymore. An example where the shortened code belongs to the same class as the original code is the following: Consider a generalized RS code with parameters $[n, k, d]_q$. For simplicity, assume that the elements $v'_1, v'_2, \ldots, v'_n$ from the definition of the generator matrix are all equal to 1. Then, according to Definition 3.18 the GRS code has the generator matrix,

$$\mathbf{G}_{GRS} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \cdots & \alpha_n^{k-1} \end{pmatrix}.$$

By now removing the first row and the first column of $\mathbf{G}_{GRS}$, the generator matrix for the shortened code then becomes,

$$\mathbf{G}_{GRS\ (shortened)} = \begin{pmatrix} \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \vdots \\ \alpha_2^{k-1} & \cdots & \alpha_n^{k-1} \end{pmatrix}.$$

The arising code is itself a GRS code with the parameters $[n-1, k-1, \geq d]_q$ and the $n-1$ code locators $\alpha_2, \ldots, \alpha_n$ as the shortened code still satisfies the definition of a GRS code.

3.9 Combining Codes

Given two existing codes, it is possible to combine them to derive a new code. In this Section, two such derived codes called *product codes* and *concatenated codes* are introduced.

3.9.1 Product Codes

A product code is a technique to make a long code by combining two codes over the same field. Given two codes $\mathcal{C}_1$ and $\mathcal{C}_2$ over the same field $\mathbb{F}_q$, a product code $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$, defined below, which is also over $\mathbb{F}_q$ can be derived. The codewords of $\mathcal{C}$ can be seen as $n_1 \times n_2$ matrices where each row is a codeword in $\mathcal{C}_1$ and each column is a codeword in $\mathcal{C}_2$. This means that the length of the product code $\mathcal{C}$ becomes $n_1 \cdot n_2$, where n_1 and n_2 denote the lengths of $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively. More formally, a product code is derived in the following way:

Definition 3.32 (Product Code)**.** *[Wac+, p. 105] Let $\mathcal{C}_1$ be an $[n_1, k_1, d_1]_q$ code and let $\mathcal{C}_2$ be an $[n_2, k_2, d_2]_q$ code. Furthermore, let $\mathbf{u}$ be a $k_1 \times k_2$ array with information symbols, represented as a matrix. The product code $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$ is obtained by encoding each row of $\mathbf{u}$ by the code $\mathcal{C}_1$ and then each column by the code $\mathcal{C}_2$.*

The following Theorem 3.33 presents the parameters of a product code.

Theorem 3.33 (Parameters of Product Code)**.** *[Wac+, p. 106] Let $\mathcal{C}_1$ be an $[n_1, k_1, d_1]_q$ code and let $\mathcal{C}_2$ be an $[n_2, k_2, d_2]_q$ code. The product code $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$ is an $[n_1 n_2, k_1 k_2, d_1 d_2]_q$ code.*

Proof. As the information vector is now a $k_1 k_2$ array which, during encoding, is extended to an $n_1 n_2$ array, the length and dimension follow by construction. The minimum distance is proved by first showing that $d \geq d_1 d_2$ and then showing that $d = d_1 d_2$:
Let $\mathbf{c}$ be a non-zero codeword of $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$. It has a non-zero row, and this row has at least d_1 non-zero elements. Each of the corresponding d_1 columns are therefore non-zero and have at least d_2 non-zero elements. Hence, $d \geq d_1 d_2$. The next step is to prove that $d = d_1 d_2$ which is done by constructing a codeword with exactly $d_1 d_2$ non-zero elements: Let $\mathbf{c}_1$ be a non-zero codeword of $\mathcal{C}_1$ of minimal weight, and let $\mathbf{c}_2$ be a non-zero codeword of $\mathcal{C}_2$ of minimal weight. Choose $\mathbf{u}_2$ (the information vector that is encoded to $\mathbf{c}_2$) such that the encoding of each row results in $\mathbf{c}_1$ in each row where $\mathbf{u}_2$ is non-zero. Encoding each column results in $\mathbf{c}_2$ in all the columns where $\mathbf{c}_1$ is non-zero. Hence, the codeword has weight $wt(\mathbf{c}_1)wt(\mathbf{c}_2) = d_1 d_2$ which proves that $d = d_1 d_2$ [RB20, p. 48]. □

It can be shown that the product code obtained by first encoding each row of $\mathbf{u}$ by $\mathcal{C}_1$ and then each column by $\mathcal{C}_2$ is the same product code as the one obtained by first encoding each column of $\mathbf{u}$ by $\mathcal{C}_2$ and then each row by $\mathcal{C}_1$. I.e. the order of encoding does not matter. Furthermore, no matter the order of encoding, then each row of every codeword of the product code is a codeword of $\mathcal{C}_1$ and each column of every codeword of the product code is a codeword of $\mathcal{C}_2$. However, this proof is left out in this thesis.

An advantage of a product code is that it is efficiently decodable as the decoders for the two existing codes are simply used. Unfortunately, product codes are in general not impressive in terms of minimum distance. Consider for example $\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{RS}(n = 255, k = 237)$. Recall that RS codes meet the Singleton bound from Theorem 3.16 which means that $d_1 = d_2 = 19$. The product code $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$ has parameters $[65025, 56169, 361]$. If $\mathcal{C}$ should meet the Singleton bound, then the minimum distance should be

$$d = n - k + 1 = 65025 - 56169 + 1 = 8857,$$

but the actual minimum distance is much smaller.

3.9.1.1 BCH/Repetition Product Code

In this Section, an example of a product code is given, and it will be investigated how many errors it can correct: Let $\mathcal{C}_{BCH}$ be a BCH code with parameters $[15, 5, \geq 7]_2$ and generator matrix,

$$\mathbf{G}_{BCH} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{F}_2^{5\times 15},$$

and let $\mathcal{C}_{REP}$ be a repetition code with parameters $[8, 1, 8]_2$ and generator matrix,

$$\mathbf{G}_{REP} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \in \mathbb{F}_2^{1\times 8}.$$

Let the $k_{REP} \times k_{BCH}$ information array be given as

$$\mathbf{u} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{F}_2^{1\times 5}.$$

In this example, the rows (or just row in this case) will be encoded first which is done using the generator matrix of the BCH code, $\mathcal{C}_{BCH}$:

$$\mathbf{c}_{BCH} = \mathbf{u} \cdot \mathbf{G}_{BCH} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{F}_2^{1\times 15}.$$

Now, each column is encoded with $\mathcal{C}_{REP}$ which means that each element is repeated $n_{REP} = 8$ times:

$$\mathbf{c} = \mathbf{G}_{REP}^T \mathbf{c}_{BCH} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{F}_2^{8 \times 15}.$$

While decoding, it is interesting to analyse the error-correction. In general, we say that a product code $\mathcal{C} = \mathcal{C}_1 \otimes \mathcal{C}_2$ can correct $\lfloor \frac{d_1 - 1}{2} \rfloor \cdot \lfloor \frac{d_2 - 1}{2} \rfloor$ errors which in this case is $\lfloor \frac{d_{BCH} - 1}{2} \rfloor \cdot \lfloor \frac{d_{REP} - 1}{2} \rfloor = \lfloor \frac{7-1}{2} \rfloor \cdot \lfloor \frac{8-1}{2} \rfloor = 3 \cdot 3 = 9$ errors. The following received word $\mathbf{r}$ shows one possible distribution of 9 errors that can be corrected (the red numbers mark the positions where an error has occurred. This means that the corresponding bit in the codeword $\mathbf{c}$ above as been flipped):

$$\mathbf{c} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{F}_2^{8 \times 15}.$$

If the decoding begins with the decoder from $\mathcal{C}_{REP}$, then the majority decision decoder will detect the errors in the first three columns, and the three columns will be correctly decoded as $[1, 0, 0]$ as the majority of the bits in each of the columns are still correct. When all columns have been decoded with the decoder from $\mathcal{C}_{REP}$, then the row can be decoded using $\mathcal{C}_{BCH}$ which will happen without failure as there are no errors left.

However, it is possible to correct a much higher number of errors if they are nicely distributed. Under the assumption that the decoder from $\mathcal{C}_{REP}$ will be used first and then the decoder from $\mathcal{C}_{BCH}$ afterwards, the following distribution of errors is the

best possible:

$$\mathbf{r} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{F}_2^{8\times 15}.$$

First, each column is decoded using the decoder from $\mathcal{C}_{REP}$, and as the majority of the bits in every column except for the first three columns is correct, then the decoder will wrongly decode the first three columns but successfully decode the rest. Luckily, the BCH code can correct $\lfloor \frac{7-1}{2} \rfloor = 3$ errrors which means that after full decoding, all 60 errors will be corrected. While it is of course unlikely to get a perfect distribution of errors randomly, it is worth noting that the error correction capability relies on the distribution of errors. Furthermore, it should also be taken into account that the decoding order can vary.

3.9.2 Concatenated Codes

A concatenated code is another technique to derive a new code by combining two codes. Concatenated codes are quite similar to product codes, but an important difference is that for a concatenated code, the two existing codes that are combined are not over the same field: Instead, a code over a big field $\mathbb{F}_{q^m}$ is combined with a code over a small field $\mathbb{F}_q$ to obtain a concatenated code over the small field $\mathbb{F}_q$. During encoding as well as decoding, the trick is to shift between the small field $\mathbb{F}_q$ and the big field $\mathbb{F}_{q^m}$. Concatenated codes are not as good at correcting single $\mathbb{F}_q$-errors as a product code, but on the other hand concatenated codes are good at dealing with a combination of single and burst errors, i.e. blocks of consecutive errors [RB20, p. 50]. The concatenated code is formally defined below:

Definition 3.34 (Concatenated Code). *[Wac+, p. 105] Let $\mathcal{C}_1$ be an $[n_1, k_1, d_1]_{q^m}$ code and let $\mathcal{C}_2$ be an $[n_2, k_2 = m, d_2]_q$ code. Note that the dimension of $\mathcal{C}_2$ is equal to the extension power of the field $\mathbb{F}_{q^m}$ of $\mathcal{C}_1$. This means that an information array $\mathbf{u}$ has size k_2k_1 over $\mathbb{F}_q$ and size k_1 over $\mathbb{F}_{q^m}$ as one element in $\mathbb{F}_{q^m}$ is represented as m elements in $\mathbb{F}_q$. The concatenated code $\mathcal{C} = \mathcal{C}_1 \circ \mathcal{C}_2$ is obtained in the following way: Represent the information array $\mathbf{u}$ as a row-vector in the extension field $\mathbb{F}_{q^m}$ which means that $\mathbf{u} \in \mathbb{F}_{q^m}^{k_1}$ and encode it using $\mathcal{C}_1$. The result is a row-vector of length n_1. Now, expand each element of the row-vector to a column-vector of length m by converting to the base field $\mathbb{F}_q$. The results is an $m \times n_1$ matrix. Finally, encode each of the n_1 columns of length m with $\mathcal{C}_2$. The result is a codeword of size $n_2 \times n_1$ over $\mathbb{F}_q$.*

The following Theorem 3.35 presents the parameters of a concatenated code.

Theorem 3.35 (Parameters of Concatenated Code)**.** *[RB20, p. 50] Let $\mathcal{C}_1$ be an $[n_1, k_1, d_1]_{q^m}$ code and let $\mathcal{C}_2$ be an $[n_2, k_2 = m, d_2]_q$ code. The concatenated code $\mathcal{C} = \mathcal{C}_1 \circ \mathcal{C}_2$ is an $[n_1 n_2, k_1 k_2, \geq d_1 d_2]_q$ code where all three parameters are expressed over $\mathbb{F}_q$.*

Proof. The proof is similar to the proof of Theorem 3.33. □

Concatenated codes usually have better parameters than product codes [Wac+, p. 110].

3.9.2.1 Reed-Solomon/Reed-Muller Concatenated Code

An example of a concatenated code is now given: Let the first code be an RS code $\mathcal{C}_{RS}$ with the parameters $[8, 5, 4]_{2^4}$ and generator matrix $\mathbf{G}_{RS} \in \mathbb{F}_{2^4}^{5\times 8}$

$$\mathbf{G}_{RS} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & z_4 & z_4^2 & z_4^3 & z_4+1 & z_4^2+z_4 & z_4^3+z_4^2 & z_4^3+z_4+1 \\ 1 & z_4^2 & z_4+1 & z_4^3+z_4^2 & z_4^2+1 & z_4^2+z_4+1 & z_4^3+z_4^2+z_4+1 & z_4^3+1 \\ 1 & z_4^3 & z_4^3+z_4^2 & z_4^3+z_4 & z_4^3+z_4^2+z_4+1 & 1 & z_4^3 & z_4^3+z_4^2 \\ 1 & z_4+1 & z_4^2+1 & z_4^3+z_4^2+z_4+1 & z_4 & z_4^2+z_4 & z_4^3+z_4 & z_4^3+z_4^2+1 \end{bmatrix},$$

and let the second code be an RM code $\mathcal{C}_{RM}$ with the parameters $[8, 4, 4]_2$ and generator matrix

$$\mathbf{G}_{RM} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \in \mathbb{F}_2^{4\times 8}.$$

Note that the extension degree of the RS extension field $\mathbb{F}_{2^4}$ indeed matches the dimension of the RM code. An information vector $\mathbf{u}$ is chosen, and in (3.10) it is viewed in the extension field:

$$\mathbf{u} = [1, 2, 3, 4, 5] \in \mathbb{F}_{2^4}^5. \tag{3.10}$$

The same information vector is in the base field $\mathbb{F}_2$ expressed in the following way, where each element of Equation (3.10) is represented as a binary column:

$$\mathbf{u} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \in \mathbb{F}_2^{4\times 5}. \tag{3.11}$$

This confirms that the dimension of the concatenated code is $k_{RM} k_{RS} = 4 \cdot 5 = 20$.

The information vector viewed in extension field as in Equation (3.10) is encrypted by multiplying with $\mathbf{G}_{RS}$, and the following is obtained:

$$\begin{aligned}\mathbf{c}_{RS} &= \mathbf{u} \cdot \mathbf{G}_{RS} \\ &= \begin{bmatrix}1 & 0 & z_4^3+z_4+1 & z_4^3+z_4^2+1 & z_4^3+z_4+1 & z_4^3+z_4^2+1 & z_4^3+z_4 & z_4^2+1\end{bmatrix} \in \mathbb{F}_{2^4}^8.\end{aligned}$$

The information vector has now been encoded using RS, however it is not fully encoded yet as the RM encoder should also be used. In order to be able to use the RM encoder, the codeword needs to be represented in the base field $\mathbb{F}_2$ instead of in the extension field $\mathbb{F}_{2^4}$. Below, the codeword $\mathbf{c}$ is represented in the base field by converting each element of $\mathbf{c}_{RS}$ to a binary column vector of length 4:

$$\mathbf{c}_{RS} = \begin{bmatrix}0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1\end{bmatrix} \in \mathbb{F}_2^{4\times 8}.$$

The final codeword for the product code is now obtained by multiplying each column with $\mathbf{G}_{RM}$ and concatenate all the codewords into one codeword. The result becomes the following codeword of length $n = n_{RS}n_{RM} = 8 \cdot 8 = 64$.

$$\mathbf{c} = \mathbf{G}_{RM}^T \mathbf{c}_{RS} = \begin{bmatrix}0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0\end{bmatrix} \in \mathbb{F}_2^{8\times 8}.$$

Decoding is done in the same manner as in the example in Section 3.9.1.1.

CHAPTER 4

The HQC Public-Key Encryption Scheme

The Hamming Quasi-Cyclic (HQC) cryptosystem was published in 2017 and is one of the alternate candidates of round 3 of the NIST standardization process. Contrary to most of the existing code-based cryptosystems such as classic McEliece, the security of HQC does not partly rely on hiding the structure of an error-correcting code. In the HQC cryptosystem, the error-correcting code that is being used is public, and the security relies on variants of the syndrome decoding problem which is an NP-hard problem.

In this section, the HQC cryptosystem will be presented. First, some necessary preliminaries are introduced. Afterwards the cryptosystem is explained, and finally the correctness, security, and known attacks are described.

4.1 Prerequisites

As in previous chapters, let $\mathbb{F}_2$ be the finite field of size 2. Let $\mathcal{V}$ be a vector space of dimension n over $\mathbb{F}_2$ for some positive $n \in \mathbb{Z}$. Let the product of $\mathbf{u}, \mathbf{v} \in \mathcal{V}$ be defined as

$$\mathbf{uv} = \mathbf{u}\ \mathrm{rot}(\mathbf{v})^T = \mathbf{v}\ \mathrm{rot}(\mathbf{u})^T = \mathbf{vu},$$

where

$$\mathrm{rot}(\mathbf{v}) = \begin{pmatrix} v_0 & v_{n-1} & \cdots & v_1 \\ v_1 & v_0 & \cdots & v_2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{n-1} & v_{n-2} & \cdots & v_0 \end{pmatrix} \in \mathbb{F}_2^{n \times n}. \tag{4.1}$$

The definition of multiplication above implies that elements of $\mathcal{V}$ can also be considered as polynomials in the ring $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$ [Agu+21a, p. 8]. In order to convert elements from $\mathbb{F}_2[X]$ to $\mathcal{R}$, they are simply reduced modulo $(X^n - 1)$.

A prime integer n is a *primitive* prime integer if the polynomial $X^n - 1/(X-1)$ is irreducible in the ring $\mathcal{R}$ [Agu+21a, p. 8].

4.2 The Public-key Encryption (PKE) Version

The Public-key Encryption Version (PKE) is IND-CPA secure which is one of two general requirements of a PKE presented in Section 2.3 (the other requirement is correctness which the PKE version of HQC also satisfies, see Section 4.3 about the correctness of HQC). As mentioned in Section 2.4, it is possible to make a transformation to obtain a KEM-DEM version that achieves IND-CCA2 which is the highest standard security requirement for a public-key cryptosystem [Agu+21a, page 15]. However, in this thesis only the PKE version will be considered.

The HQC encryption scheme uses two different codes [Sch+20, p. 3]:

- A public code $\mathcal{C}$ of length n and dimension k. For this code, it is assumed that there are both an efficient encoding algorithm and an efficient decoding algorithm, both publicly known.
- The second code is a random binary code of length $2n$ with a parity-check matrix defined as $(I, rot(\mathbf{h})) \in \mathbb{F}_2^{n \times 2n}$ where $\mathbf{h}$ is a vector of length n generated uniformly at random during the key generation, and I is the $n \times n$ identity matrix. It is assumed that no one knows an efficient decoding algorithm for the second code.

In this Chapter, it will not be covered how the public code $\mathcal{C}$ (the first code mentioned above) is chosen. This will instead be covered in Chapter 6 in which the choice of $\mathcal{C}$ will be thoroughly investigated. The second mentioned code is a random code, a concept which is defined below:

Definition 4.1 (Random Binary Linear Code)**.** *[MO15, p. 207] A code $\mathcal{C}$ is a random binary linear code if it is generated by a matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$ of which the entries are chosen independently and uniformly at random from $\mathbb{F}_2$ with the restriction that $\mathbf{G}$ has rank k. It follows that the entries of the corresponding parity-check matrix $\mathbf{H}$ are also independently and uniformly at random distributed in $\mathbb{F}_2$.*

The second code is not only a random code, it is also of the type called *quasi-cyclic* codes. The benefit of using quasi-cyclic codes is that the public key size kan be reduced significantly, see Section 4.6.

Definition 4.2 (Quasi-Cyclic Codes)**.** *[Agu+21a, p. 10] Let $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{s-1}) \in \mathbb{F}_2^{sn}$ where $\mathbf{c}_i \in \mathbb{F}_2^n$ for $i = 0, \ldots, s-1$, for some $s \in \mathbb{N}_{>0}$. An $[sn, k, d]_2$ linear code $\mathcal{C}$*

is Quasi-Cyclic of index s if for any $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{s-1}) \in \mathcal{C}$ a simultaneous circular shift as in Definition 3.11 to all $\mathbf{c}_0, \ldots, \mathbf{c}_{s-1}$ is also a codeword in $\mathcal{C}$. In other words, by considering each block $\mathbf{c}_i$ as a polynomial in the ring $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$, the code $\mathcal{C}$ is Quasi-Cyclic of index s if for any $\mathbf{c} = (\mathbf{c}_0, \ldots, \mathbf{c}_{s-1}) \in \mathcal{C}$ it holds that also $(X \cdot \mathbf{c}_0, \ldots, X \cdot \mathbf{c}_{s-1}) \in \mathcal{C}$.

The random code is a quasi-cyclic code of index $s = 2$ because it has length $2n$.

As mentioned in Section 2.3, a public-key encryption scheme consists of the three algorithms: Key generation, Encryption, and Decryption. The three algorithms in HQC are presented in Algorithms 3, 4, and 5 below.

In the setup fase before the keys are generated, some parameters are chosen: A code $\mathcal{C}$ of length $n \approx n_1 n_2$, dimension k, and error correction capability δ. Note that δ is the number of errors that we at least expect to be able to correct. The code $\mathcal{C}$ is a product or concatenated code of two codes of lengths n_1 and n_2 respectively. In order to avoid algebraic attacks (see Section 4.5), n is chosen as the smallest primitive prime greater than $n_1 n_2$ [Agu+21a, p. 16]. Furthermore, the weight parameters ω, $\omega_{\mathbf{r}}$, and $\omega_{\mathbf{e}}$, used to predetermine the Hamming weight of randomly generated vectors, are chosen. Note that the inputs to the Key Generation Algorithm 3 should be seen as inputs to Algorithms 4 and 5 as well.

Algorithm 3 Key Generation [Agu+21a, p. 16]

1: **Input:** param $= (n, k, \delta, \omega, \omega_{\mathbf{r}}, \omega_{\mathbf{e}})$
2: **Output:** public key pk $= (\mathbf{h}, \mathbf{s})$ and secret key sk $= (\mathbf{x}, \mathbf{y})$
3: $\mathbf{h} \xleftarrow{\$} \mathcal{R}$ ▷ choose uniformly at random
4: $(\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}^2$ such that $wt(\mathbf{x}) = wt(\mathbf{y}) = \omega$ ▷ choose uniformly at random
5: $\mathbf{s} \leftarrow \mathbf{x} + \mathbf{hy}$
6: **return** pk $= (\mathbf{h}, \mathbf{s})$, sk $= (\mathbf{x}, \mathbf{y})$

In Algorithm 3, both $\mathbf{h}$, $\mathbf{x}$, and $\mathbf{y}$ are chosen uniformly at random, but with the requirement that both $\mathbf{x}$ and $\mathbf{y}$ have Hamming weight ω. Such a requirement does not exist for $\mathbf{h}$. Note that the public key *pk* consists of the two ingredients for an instance of the syndrome decoding problem, i.e. a parity-check matrix represented as the vector $\mathbf{h}$ and a syndrome $\mathbf{s}$. It follows from Algorithm 3 that retrieving the secret key *sk* from the public key *pk* requires solving an instance of the syndrome decoding problem:

$$\mathbf{s} = \mathbf{x} + \mathbf{hy} = \begin{pmatrix} \mathbf{x} & \mathbf{y} \end{pmatrix} \begin{pmatrix} \mathbf{I} \\ \mathrm{rot}(\mathbf{h})^T \end{pmatrix} = \mathbf{eH}^T.$$

Algorithm 4 Encryption [Agu+21a, p. 16]

Input: public key $\text{pk} = (\mathbf{h}, \mathbf{s})$ and plaintext $\text{pt} = \mathbf{m}$
Output: ciphertext $\text{ct} = (\mathbf{u}, \mathbf{v})$
$\mathbf{e}' \xleftarrow{\$} \mathcal{R}$ such that $wt(\mathbf{e}') = \omega_{\mathbf{e}}$ ▷ choose uniformly at random
$(\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathcal{R}^2$ such that $wt(\mathbf{r}_1) = wt(\mathbf{r}_2) = \omega_{\mathbf{r}}$ ▷ choose uniformly at random
$\mathbf{u} \leftarrow \mathbf{r}_1 + \mathbf{h}\mathbf{r}_2$
$\mathbf{v} \leftarrow \mathcal{C}.\text{Encode}(\mathbf{m}) + \mathbf{s}\mathbf{r}_2 + \mathbf{e}'$
return $\text{ct} = (\mathbf{u}, \mathbf{v})$

In the Encryption algorithm, $\mathbf{e}'$, $\mathbf{r}_1$, and $\mathbf{r}_2$ are chosen uniformly at random, but with the requirement that $\mathbf{e}'$ has Hamming weight ω_e, and $\mathbf{r}_1$ and $\mathbf{r}_2$ both have Hamming weight ω_r. Furthermore, another instance of the syndrome decoding problem appears, this time in order to retrieve $\mathbf{r}_1$ and $\mathbf{r}_2$:

$$\mathbf{u} = \mathbf{r}_1 + \mathbf{h}\mathbf{r}_2 = \begin{pmatrix} \mathbf{r}_1 & \mathbf{r}_2 \end{pmatrix} \begin{pmatrix} \mathbf{I} \\ \text{rot}(\mathbf{h})^T \end{pmatrix} = \tilde{\mathbf{e}}\mathbf{H}^T.$$

It follows from Algorithm 4 that

$$\begin{aligned} \mathbf{v} &= \mathcal{C}.\text{Encode}(\mathbf{m}) + \mathbf{s}\mathbf{r}_2 + \mathbf{e}' \\ &= \mathcal{C}.\text{Encode}(\mathbf{m}) + (\mathbf{x} + \mathbf{h}\mathbf{y})\mathbf{r}_2 + \mathbf{e}' \\ &= \mathcal{C}.\text{Encode}(\mathbf{m}) + \mathbf{x}\mathbf{r}_2 + \mathbf{h}\mathbf{y}\mathbf{r}_2 + \mathbf{e}', \end{aligned}$$

and as long as the weight of $\mathbf{h}$ is not too small, then the vector,

$$\mathbf{s}\mathbf{r}_2 + \mathbf{e}' = \mathbf{x}\mathbf{r}_2 + \mathbf{h}\mathbf{y}\mathbf{r}_2 + \mathbf{e}',$$

also has large weight. Hence, $\mathbf{v}$ can be seen as an erroneous codeword of $\mathcal{C}$ with an error of weight $wt(\mathbf{s}\mathbf{r}_2 + \mathbf{e}')$ which is hopefully large. If the error is large enough, then the erroneous codeword $\mathbf{v}$ cannot be efficiently decoded, and hence the ciphertext cannot be decoded without the secret key.

Algorithm 5 Decryption [Agu+21a, p. 16]

Input: secret key $\text{sk} = (\mathbf{x}, \mathbf{y})$ and ciphertext $\text{ct} = (\mathbf{u}, \mathbf{v})$
Output: decrypted message $\mathbf{m}$
$\mathbf{v}' \leftarrow \mathbf{v} - \mathbf{u}\mathbf{y}$
$\mathbf{m} \leftarrow \mathcal{C}.\text{Decode}(\mathbf{v}')$
return $\mathbf{m}$

From Algorithm 5, it follows that

$$\begin{aligned}\mathbf{v}' &= \mathbf{v} - \mathbf{uy} \\ &= (\mathcal{C}.\text{Encode}(\mathbf{m}) + \mathbf{sr}_2 + \mathbf{e}') - (\mathbf{r}_1 + \mathbf{hr_2})\mathbf{y} \\ &= (\mathcal{C}.\text{Encode}(\mathbf{m}) + (\mathbf{x} + \mathbf{hy})\mathbf{r}_2 + \mathbf{e}') - (\mathbf{r}_1 + \mathbf{hr_2})\mathbf{y} \\ &= \mathcal{C}.\text{Encode}(\mathbf{m}) + \mathbf{xr}_2 + \mathbf{hyr}_2 + \mathbf{e}' - \mathbf{r}_1\mathbf{y} - \mathbf{hr}_2\mathbf{y} \\ &= \mathcal{C}.\text{Encode}(\mathbf{m}) + \mathbf{xr}_2 + \mathbf{e}' - \mathbf{r}_1\mathbf{y},\end{aligned}$$

is an erroneous codeword of $\mathcal{C}$ with an error $\mathbf{xr}_2 + \mathbf{e}' - \mathbf{r}_1\mathbf{y}$ of small weight, since $\mathbf{x}, \mathbf{y}, \mathbf{r}_1, \mathbf{r}_2$, and $\mathbf{e}'$ have small weights. For reference, the current version of HQC (for security category I, see Chapter 6) uses $\omega_r = \omega_e = 72$ and $\omega = 65$ to denote a small weight for $n = 17669$ ($n_1 n_2 = 17664$) [Agu+21b].

4.3 Correctness

The correctness of the HQC encryption scheme relies on the decoding capability of the code $\mathcal{C}$. Since the error of $\mathbf{v}'$ is $\mathbf{e}_{\mathbf{v}'} = \mathbf{xr}_2 + \mathbf{e}' - \mathbf{r}_1\mathbf{y}$ as explained above, $\mathcal{C}$.Decode correctly decodes $\mathbf{v}'$ whenever [Agu+21a, p. 18]

$$wt(\mathbf{e}_{\mathbf{v}'}) \leq \delta, \tag{4.2}$$

where δ is the number of errors that $\mathcal{C}$ can correct at least. In order to compute the probability of successful decoding, i.e. the probability of satisfying Equation (4.2), it is necessary to understand the probability distribution of the weight of the error vector $\mathbf{e}_{\mathbf{v}'}$. In [Agu+21a, p. 18], there is an analysis of the probability distribution of the weight of $\mathbf{e}_{\mathbf{v}'}$ in which the assumption that each coordinate of $\mathbf{e}_{\mathbf{v}'}$ is an independent variable, is made. Using this assumption, it is possible to interpret the weight distribution of $\mathbf{e}_{\mathbf{v}'}$ as a binomial distribution with a probability parameter p. In [Agu+21a, p. 21], they also present a graph showing a comparison between the weight of $\mathbf{e}_{\mathbf{v}'}$ generated using HQC-128 (the version for security category I, see Chapter 6) and its binomial approximation. Unfortunately, the graph shows that the assumption that each coordinate of $\mathbf{e}_{\mathbf{v}'}$ is independent does not exactly hold, and therefore only an approximate decoding probability is known.

4.4 Security of HQC

The security of HQC relies on the hardness of the syndrome decoding problem. Recall that

$$\mathbf{s} = \mathbf{x} + \mathbf{hy} = \begin{pmatrix}\mathbf{x} & \mathbf{y}\end{pmatrix} \begin{pmatrix}\mathbf{I} \\ \text{rot}(\mathbf{h})^T\end{pmatrix} = \mathbf{eH}^T,$$

where $pk = (\mathbf{h}, \mathbf{s})$ is the public key, and $sk = (\mathbf{x}, \mathbf{y})$ is the secret key. The public key provides the parity-check matrix and the syndrome, and as long as the syndrome decoding problem is hard, the secret key is safe. Exactly the same holds for the instance of the syndrome decoding proble from the Encryption Algorithm 4:

$$\mathbf{u} = \mathbf{r}_1 + \mathbf{h}\mathbf{r}_2 = \begin{pmatrix} \mathbf{r}_1 & \mathbf{r}_2 \end{pmatrix} \begin{pmatrix} \mathbf{I} \\ \mathrm{rot}(\mathbf{h})^T \end{pmatrix} = \tilde{\mathbf{e}}\mathbf{H}^T.$$

Recall from Chapter 2 that the minimal criteria for a public-key encryption scheme is that it is IND-CPA secure, i.e. that an adversary should not be able to efficiently guess which plaintext has been encrypted even if he knows it is one among two plaintexts of his choice. This criteria is satisfied from the above arguments.

As mentioned, it is possible to make a transformation of the scheme to obtain a KEM version that achieves *indistinguishability against adaptive chosen-ciphertext attacks (IND-CCA2)* which is the highest standard security requirement for a public-key cryptosystem [Agu+21a, p. 15]. However, this part will not be covered in this thesis.

4.5 Known Attacks

There are some attacks on the form of the polynomial generating the cyclic structure, i.e. the polynomial $x^n - 1$ which is used to generate the ring $\mathcal{R} = \mathbb{F}_2[x]/(x^n - 1)$. Without going into details, it will briefly be explained that this is the reason why n is chosen as the smallest primitive prime integer greater than $n_1 n_2$ [Agu+21a, p. 43]. According to [Agu+21a, p. 43], such attacks are especially efficient when the polynomial $x^n - 1$ has many low degree factors, and in order to make such attacks inefficient, it is ensured that $x^n - 1$ has only the two irreducible factors, $x - 1$ and $x^{n-1} + x^{n-2} + \cdots + x + 1$. This is achieved by choosing n as the smallest primitive prime integer greater than $n_1 n_2$.

There is another important class of attacks: Recall the syndrome decoding problem from Chapter 3. The best known algorithms to solve the syndrome decoding problem belong to a class of algorithms known as *information set decoding (ISD)*, initially discovered by Prange in 1962 [Pra62]. This means that ISD algorithms play an important part in the list of known attacks against code-based cryptosystems such as HQC. In this Section, Prange's ISD algorithm is presented. Some of the researchers that have later improved Prange's algorithm are the following: Lee and Brickell in 1988 [LB88], Stern in 1989 [Ste89], Dumer in 1996 [Dum96], Both and May in 2018 [BM18], and recently Bellini and Esser in 2021 [EB21a].

4.5.1 Prange's Information Set Decoding

The goal in Prange's ISD algorithm is to guess k error-free positions and use them to retrieve the message from the codeword. Note that in general the following holds:

Given a codeword $\mathbf{c} \in \mathbb{F}_q^n$ and generator matrix $\mathbf{G} \in \mathbb{F}_q^{k \times n}$, the corresponding message $\mathbf{m} \in \mathbb{F}_q^k$ can be retrieved in the following way: Choose k independent columns of $\mathbf{G}$, and let $\hat{\mathbf{G}}$ be a matrix containing only these k columns. Then:

$$\mathbf{m} = \mathbf{c} \cdot \hat{\mathbf{G}}^{-1}.$$

Prange's ISD algorithm can then be described by the following steps [WP21, p. 41]: Let $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ be a generator matrix, $\mathbf{m} \in \mathbb{F}_q^k$ a message, and $\mathbf{c} \in \mathbb{F}_q^n$ the corresponding codeword. Then:

1. Choose a random set $\mathcal{I}$ of size k, with distinct elements such that $\mathcal{I} \subset \{0, \ldots, n-1\}$, $|\mathcal{I}| = k$.
2. Construct a submatrix $\mathbf{G}^{\mathcal{I}}$ of the generator matrix $\mathbf{G}$ consisting of the k columns indexed by $\mathcal{I}$.
3. Denote

   $$\mathbf{c}_{\mathcal{I}} = \mathbf{m} \cdot \mathbf{G}^{\mathcal{I}} + \mathbf{e}_{\mathcal{I}}.$$

   If we are lucky and $wt(\mathbf{e}_{\mathcal{I}}) = 0$, then the message $\mathbf{m}$ can be decoded exactly by

   $$\mathbf{m} = \mathbf{c}_{\mathcal{I}} \cdot (\mathbf{G}^{\mathcal{I}})^{-1}.$$

The ISD algorithm can formally be stated as:

Algorithm 6 Prange's Information Set Decoding [WP21, p. 41]

1: **Input:** generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$, codeword $\mathbf{c} \in \mathbb{F}_2^n$, and a number of errors t
2: **Output:** the message $\mathbf{m} \in \mathbb{F}_2^k$
3: choose a set $\mathcal{I}$ of distinct elements such that $\mathcal{I} \subset \{0, \ldots, n-1\}$, $|\mathcal{I}| = k$, at random
4: $\mathbf{c}_{\mathcal{I}} \leftarrow \mathbf{m} \cdot \mathbf{G}^{\mathcal{I}} + \mathbf{e}_{\mathcal{I}}$
5: $\hat{\mathbf{m}} \leftarrow \mathbf{c}_{\mathcal{I}} \cdot (\mathbf{G}^{\mathcal{I}})^{-1}$
6: if $wt(\hat{\mathbf{m}} \cdot \mathbf{G} - \mathbf{c}) = t$ **do:**
7: $\quad \mathbf{m} \leftarrow \hat{\mathbf{m}}$
8: else **do:**
9: $\quad$ go to line 3
10: **return m**

Hence, only if the positions in $\mathcal{I}$ are error-free, then the message $\mathbf{m}$ can be retrieved. An important question is what the probability of getting only error-free positions in

$\mathcal{I}$ is. If t errors occurred, then there are $\binom{n-t}{k}$ ways of choosing k error-free positions which means that the probability becomes

$$\frac{\binom{n-t}{k}}{\binom{n}{k}}.$$

In order to use Prange's ISD algorithm for measurements and comparisons, a factor known as Prange's *work factor* can be used and is defined as [WP21, p. 42]:

$$WF = \left(\frac{\binom{n-t}{k}}{\binom{n}{k}}\right)^{-1} \cdot k^3,$$

where k^3 is the work per iteration.

4.6 Advantages and Disadvantages

Some of the advantages of the HQC cryptosystem are [Agu+21a, p. 43-44]:

- It relies on a well-understood NP-hard problem: The Syndrome Decoding Problem.
- It does not rely on hiding the structure of the code being used in contrast to many other code-based cryptosystems such as Classic McEliece [Sch+20, p. 1]
- Small public key size: The public key $pk = (\mathbf{h}, \mathbf{s})$ consists of the two ingredients used in an instance of the syndrome decoding problem: The parity-check matrix, or a representation of it in terms of the vector $\mathbf{h}$ and the syndrome $\mathbf{s}$. The fact that the quasi-cyclic structure is used makes it possible to represent the full parity-check matrix by only the vector $\mathbf{h}$ of length n instead of storing a full matrix. Hence, the public key size in the HQC cryptosystem is much smaller compared to the public key size in other code-based cryptosystems such as Classic McEliece.
- Close estimations of decryption failure rate.
- Efficient implementations.

Some of the disadvantages are [Agu+21a, p. 44]:

- As explained in Section 4.3, the analysis of the probability of successful decoding still needs some improvements.

- Low encryption rate: If we want to encrypt messages with more than 128 or 192 or 256 bits, depending on the chosen security category, see Chapter 6, then it is necessary to either split the message in chunks of size 128, 192, or 256, and run multiple instances of HQC, or to define a new set of parameters $(n, n_1, n_2, k, \omega, \omega_e, \omega_r, \delta)$ which allows for the preferred length of the message. In other words, the encryption scheme is not flexible in terms of changing the number of bits in the message that is encrypted [Agu+21a, p. 44].

CHAPTER 5
Implementation

The error-correcting codes that were presented in Chapter 3 have been implemented in the python library, SageMath. Furthermore, implementations of the product code and concatenated code constructions have been developed such that they take two error-correcting codes as input and construct the derived code. Finally, the HQC encryption scheme has been implemented. All the implementations can be found in the Github repository `https://github.com/AmalieDue/HQC`.

The Github repository contains two folders, `notebooks` and `py files`. The two folders contain the exact same files, but in `notebooks` they are in Jupyter Notebook format, and in `py files` the notebooks are converted to `.py` files for use on the DTU HPC cluster hardware.

The implementation has been structured as python classes in the following way:

- One class for each code: `RSCode`, `RMCode`, `BCHCode`, and `RepetitionCode`.
- One class for each type of derived code: `ConcatenatedCode` and `ProductCode`. Each class takes two instances of error-correcting code classes as input.
- One class for HQC: `HQC`. This class takes an instance of either `ConcatenatedCode` or `ProductCode` as input. It can also just take an instance of one of the error-correcting code classes as input which has been added as a part of the testing, see Chapter 6.

5.1 Implementation of Error-Correcting Codes

In this section, the implementation of each error-correcting code will be described. First, the general structure which is the same for all four codes is described:

When using a class, the first step is to define an instance of the chosen class with its necessary parameters. Below is a generic example, where "`X`" can be replaced directly by either `RS, RM, BCH`, or `Repetition`.

```
C = XCode(parameter1, parameter2, ...)
```

Listing 5.1: Generic example of defining an instance of a code class.

Each code class can handle the following three different types of input:

```
m = [1,2,3,4,5,6,7] #list of integers
m = '101010110111010' #bit string
m = [1, z4, z4 + 1, z4^2, z4^2 + 1] #list of elements from finite field
                                    #in this case from GF(2**4)
```

Listing 5.2: Data types that can be handled.

Each code class has an `Encoding` and `Decoding` function which are called in the following way:

```
m = '.....' #some message

c = C.Encoding(m, out = 'pol') #codeword is saved in c

d = C.Decoding(c, out = 'int') #decoded word is saved in d
```

Listing 5.3: Encoding and Decoding functions.

The `Encoding` and `Decoding` functions both take an input `out`. This input can be used to specify which data type the output should have. Possible values for the `out` input are: `'int'`, i.e. integer representation, `'bin'`, i.e. bit string representation, and `'pol'`, i.e. polynomial representation, corresponding to the three different types of input defined above.

The encoding and decoding algorithms implemented are the ones presented in Chapter 3.

5.1.1 Reed-Solomon Code

In the implementation of the RS code, the generator matrix is constructed as in Definition 3.18 with

$$v_i' = 1 \quad \text{for } i = 1, \ldots, n.$$

The code locators have been chosen in the following way: Let $\mathbb{F}_{q^m}$ be the RS field, and let α be a primitive element in $\mathbb{F}_{q^m}$. Then the code locators are chosen as

$$\alpha_i = \alpha^{i-1} \quad \text{for } i = 1, \ldots, n.$$

In Listing 5.4, the inputs to the RS code are shown, and they are given example values.

```
C = RSCode(n=9, k=5, q=2**4, shortening = 0)
```

Listing 5.4: An instance of the RS class with example parameters.

The RS class takes the following inputs, with some being optional:

- The length `n`.
- The dimension `k`. (The minimum distance is derived from `n` and `k` and is therefore not given as input).
- The field size `q`.
- (Optional) The amount of `shortening`: The RS code is shortened by the input value. If no value is input, the instance defaults to 0, i.e. no shortening.

An example of encoding and decoding with this class is given below. There will not be examples for the other error-correcting code classes as they follow the same procedure.

First, encoding:

```
    C = RSCode(n=9, k=5, q=2**4, shortening = 0)

    m = [1,2,3,4,5,6,7] #list of integers

    c = C.Encoding(m, out = 'pol')
    print('codeword:', c)

Out[]: codeword:  (1, 0, z4^3 + z4 + 1, z4^3 + z4^2 + 1, z4^3 + z4 + 1, z4^3 +
    z4^2 + 1, z4^3 + z4, z4^2 + 1, z4^2, 1, z4^3, z4^3 + 1, z4^3 + z4 + 1, z4^3
     + z4^2 + z4 + 1, z4^2 + z4 + 1, z4^2, z4, z4^3 + z4^2 + z4)
```

Listing 5.5: Encoding with RSCode class.

Then, decoding:

```
        d = C.Encoding(c, out = 'pol')
        print('decoded word:', d)

    Out[]: decoded word:  [1,2,3,4,5,6,7,0,0,0]
```

Listing 5.6: Decoding with RSCode class.

Note that the decoded word is the message plus some zero padding. This happens if the length of the message is not a multiple of the dimension `k`. The message is then zero padded in order to get a multiple of `k`.

5.1.2 Reed-Muller Code

In Listing 5.7, the inputs to the RM class are shown, and they are given example values.

```
C = RMCode(r = 1, m = 4, q = 2)
```

Listing 5.7: An instance of the RMCode class.

The inputs to the RM class are:

- The order `r`.
- The number of variables `m`.
- The field size `q`. However, current implementation only supports $\texttt{q} = 2$.

5.1.3 BCH Code

In Listing 5.8, the inputs to the RM class are shown, and they are given example values.

```
C = BCHCode(n = 31, b = 1, D = 11, q = 2, shortening = 0)
```

Listing 5.8: An instance of the BCHCode class.

The inputs to the BCH class are:

- The length `n`.
- The parameter `b`.
- The minimum distance of the underlying RS code `D`.
- The field size `q`. However, current implementation only supports $\texttt{q} = 2$.
- (Optional) The amount of `shortening`: The BCH code is shortened by the input value. If no value is input, the instance defaults to 0, i.e. no shortening.

As the BCH code is a subfield subcode of an RS code, the decoder from the RS class is used for decoding the BCH code. This means that an instance of the `RSCode` class is defined inside the `BCHCode` class such that the RS decoding algorithm can be used. However, this is done automatically, so the user can just use the BCH as usual with the `Encoding` and `Decoding` functions.

5.1.4 Repetition Code

In Listing 5.9, the inputs to the RM class are shown, and they are given example values.

```
C = RepetitionCode(n = 5, q = 2)
```

Listing 5.9: An instance of the RepetitionCode class.

The inputs to the Repetition class are:

- The length `n`.
- The field size `q`. However, current implementation only supports $q = 2$.

5.2 Implementation of Derived Codes

A `ProductCode` class and a `ConcatenatedCode` class have been implemented. An instance of the `ProductCode` class is initialized as

```
        C = ProductCode(code1, code2)
```

Listing 5.10: An instance of the ProductCode class.

and similarly for the concatenated code:

```
        C = ConcatenatedCode(code1, code2)
```

Listing 5.11: An instance of the ConcatenatedCode class.

where code1 and code2 both are an instance of one of the error-correcting code classes. The `ProductCode` and `ConcatenatedCode` class both have `Encoding` and `Decoding` functions, exactly as the previous classes.

An example with a product code is given below:

```
        C = ProductCode(RSCode(n=9, k=5, q=2**4), RSCode(n=7, k=3, q=2**4))

        m = '11001100110011001100' #the message is initialized

        c = C.Encoding(m, out = 'bin')
        print("codeword: ", c)

    Out []: codeword :  1100111110000000000100100000010111010
```

Listing 5.12: Encoding with the ProductCode class.

and then decoding:

```
        d = C.Decoding(c, out = 'bin')
        print("decoded word:", d)

    Out[]: decoded word :  11001100110011001100
```

Listing 5.13: Decoding with the ProductCode class.

5.3 Implementation of HQC

Finally, the HQC cryptosystem has been implemented. An instance of the HQC class is initialized as

```
        Code1 = BCHCode(n = 1023, b = 1, D = 115, q = 2, shortening = 257)
        Code2 = RepetitionCode(n = 31, q = 2)
        HQC_Code = ProductCode(Code1,Code2)

        HQC = HQC(w = 65, w_e = 72, w_r = 72, C = HQC_Code, key_type = 'pol',
            simulation = False, single_code = False)
```

Listing 5.14: An instance of the HQC class.

The inputs to the HQC class are:

- The weight parameters, `w, w_e`, and `w_r`.
- The underlying error-correcting code `C`.
- The data type of the public key and secret key, `key_type`. Possible values are: `'pol', 'int'`, and `'bin'` as usual. The default value is `'pol'` as this is the most efficient data type for the implementation.
- A parameter `simulation` which specifies whether simulation tests are run. Possible values are `True` and `False`, with `False` being the default. If `True` multiple tests detailed in Chapter 6 are performed, if `False` no tests are performed.
- A parameter `single_code`. Possible values are `True` and `False`, default is `False`. If `True` is given, the implementation knows that it should handle `C` as a single code instead of a product code or a concatenated code. This parameter has been added as a part of the testing and analysis.

There are three functions in the HQC class, `KeyGen`, `Encrypt`, and `Decrypt`. However, only the `Encrypt` and `Decrypt` functions should be used by the user. The `KeyGen` is automatically called when an instance of the HQC class is initialized such that the key generation is a part of the setup fase.

Find below an example of how to use the `HQC` class:

```
        HQC = HQC(w = 3, w_e = 3, w_r = 3, C = ConcatenatedCode(RSCode(n = 15,
            k = 7, q = 2**4), RMCode(r = 1, m = 3, q = 2)), key_type = 'pol',
            simulation = False, single_code = False)

        print('Public key: ', C.pk)
        print('Secret key: ', C.sk)

    Out []: Public key:  (a^119 + a^118 + a^117 + a^116 + a^115 + a^112 + a^111
             + a^110 + a^109 + a^107 + a^105 + a^104 + a^102 + a^101 + a^99 + a^97
            + a^95 + a^92 + a^91 + a^90 + a^88 + a^87 + a^86 + a^83 + a^81 + a^79 +
             a^77 + a^76 + a^75 + a^74 + a^73 + a^71 + a^69 + a^68 + a^67 + a^65 +
            a^63 + a^58 + a^55 + a^54 + a^50 + a^47 + a^46 + a^44 + a^41 + a^40 + a
            ^35 + a^33 + a^31 + a^29 + a^28 + a^25 + a^24 + a^20 + a^19 + a^18 + a
            ^17 + a^16 + a^14 + a^13 + a^8 + a^7 + a^4 + a^3 + 1, a^118 + a^115 + a
            ^113 + a^111 + a^107 + a^99 + a^97 + a^96 + a^95 + a^94 + a^90 + a^87 +
             a^84 + a^83 + a^82 + a^80 + a^79 + a^77 + a^74 + a^71 + a^68 + a^65 +
            a^64 + a^63 + a^60 + a^59 + a^58 + a^57 + a^54 + a^52 + a^51 + a^49 + a
            ^41 + a^40 + a^39 + a^37 + a^36 + a^33 + a^32 + a^31 + a^30 + a^29 + a
            ^28 + a^26 + a^24 + a^21 + a^20 + a^19 + a^18 + a^16 + a^14 + a^12 + a
            ^9 + a^7 + a^3 + a^2 + a + 1)

        Secret key:  (a^116 + a^51 + a, a^88 + a^56 + a^13)
```

Listing 5.15: An example initialization of the HQC class.

Encryption using the initialized HQC class:

```
        m = '10101010101010101010101010'

        c = C.Encrypt(m, out = 'bin')
        print('ciphertext:', c)

    Out []:  ciphertext: ('
        00110100101010111110010101101111101010111000001010100011000101011
    000110001111001010111010111111011000011001010000010000010', '
        00101111000010101110001101001110100101001100011001000001011011000 1
    00110001011010010001101011001110000100011100000011010 1')
```

Listing 5.16: Encryption with the HQC class.

Decrypting the ciphertext:

```
        d = C.Decrypt(c, out = 'bin')
        print('plaintext:', d)

    Out []:  plaintext: 10101010101010101010101010
```

Listing 5.17: Decryption with the HQC class.

As seen the resulting plaintext matches the message `m`.

CHAPTER 6

A Better Choice of $\mathcal{C}$?

Notice: Unfortunately, late in the process it was spotted that the way $\mathbf{h}$ is generated in the SageMath implementation is different from the way it is generated in the `C` code submitted to NIST [Agu+21b]. In practice, it means that a $\mathbf{h}$ generated with the SageMath code will have any Hamming weight between 0 and n will equal probability whereas a $\mathbf{h}$ generated with the `C` code always will have Hammign weight around $\frac{n}{2}$ for large n. However, it does not have an effect on the security level estimates presented in this Chapter.

In this Chapter, the choice of the public code $\mathcal{C}$ that is used in HQC will be investigated. Recall that there should exist both a public efficient encoding algorithm and a public efficient decoding algorithm for $\mathcal{C}$. One thing is to choose a code class for $\mathcal{C}$, another thing is to choose a set of parameters for $\mathcal{C}$ that makes the encryption scheme secure. The security level is measured as an estimate of the attack cost required to break the encryption scheme given the chosen set of parameters [EB21b, p. 1]. One such estimation is Prange's workfactor as described in Section 4.5.1, but many similar algorithms with improvements have later been developed. In order to compute such estimations, the syndrome decoding estimator in [BE21] will be used.

A way to compare the different submissions to the NIST standardization process is to divide them into different security categories. NIST has defined 5 different categories, but only category I, III, and V are of interest in this context [20221]:

Category	Security description
I	At least as hard to break as AES128 (exhaustive key search)
III	At least as hard to break as AES192 (exhaustive key search)
V	At least as hard to break as AES256 (exhaustive key search)

Table 6.1: Security Categories [20221].

A KEM scheme that aims to match security category I should have $k \geq 128$ such that an AES128-key can be exchanged (recall the definition of a KEM scheme in Section 2.4). Similarly, a KEM scheme that aims to match security category III and V, respectively, should have $k \geq 192$ and $k \geq 256$, respectively.

This chapter will focus on security category I.

In the first version of HQC submitted to NIST in 2017, the code $\mathcal{C}$ was chosen as a product code of a shortened BCH code and a repetition code with the following parameters for security category I (taken from the implementation package from 2020/05/29 [Agu+21b]):

$$\begin{aligned}&\mathcal{C}_{BCH} = [766, 256, 115]_2, \;\; \mathcal{C}_{Rep} = [31, 1, 31]_2,\\ &\mathcal{C} = \mathcal{C}_{BCH} \otimes \mathcal{C}_{Rep} = [23869, 256, 3751]_2, \;\; (n_1 n_2 = 23746)\\ &\omega = 67, \;\; \omega_r = \omega_r = 77.\end{aligned}$$

The security level estimates can be calculated using the syndrome decoding estimator in [BE21] in the following way: The estimator takes three inputs, n, k, and w. Note that the inputs refer to the parameters of the random code and not the parameters of $\mathcal{C}$. Recall from the beginning of Chapter 4 that the random code has length $2n$ and dimension n (the dimension could change a tiny bit if the parity-check matrix turned out to not have full rank, but for simplicity it is assumed that the dimension is n) which follows from the size of its parity-check matrix. Furthermore, the input parameter w should be given the value 2ω since the random code is a quasi-cyclic code with index $s = 2$. Hence, in order to compute the security level estimates for the HQC product code the following is computed:

```
sd_estimate_display(n=47738,k=23869,w=154)
```

Listing 6.1: Usage of the syndrome decoding estimator in [BE21].

The syndrome decoding estimator outputs many different estimates, but only the three following estimates will be presented throughout this Chapter: Prange from 1962 [Pra62], Stern from 1989 [Ste89], and Both and May from 2018 [BM18]. Note that throughout this Chapter, Both and May will be referred to as BM in all tables.

Table 6.2 shows the output from the function call in Listing 6.1. This is the security level estimates of the HQC product code of a shortened BCH code and a repetition code calculated using the three estimates mentioned above, i.e. Prange, Stern, and Both and May.

	Category I ($2n = 47738$)
Prange	196.6
Stern	174.8
Both-May (BM)	175.2

Table 6.2: The security level estimates computed using the syndrome decoding estimator [BE21] for the product code of a shortened BCH code and a repetition code [Agu+21b].

In the HQC updates from May 2020, the submitters introduced a new choice of $\mathcal{C}$, a concatenated code of a shortened RS code and a duplicated RM code. This thesis does not cover duplicated RM codes, therefore [Agu+21a, p. 25] should be used for further details. In the HQC updates from October 2020, the submitters announced that the concatenated code of a shortened RS code and a duplicated RM code is strictly better than the product code of a shortened BCH code and a repetition code, and therefore the product code would not be used anymore [Agu+21a, p. 3]. The change of $\mathcal{C}$ did not change the encryption scheme, but it made it possible to decrease the size of the public key by 17% while keeping an acceptable security level [Agu+21a, p. 4]. In the newest version of HQC from 2021/06/06, the parameters for security category I are as follows [Agu+21b]:

$$\begin{aligned}
&\mathcal{C}_{RS} = [46, 16, 31]_2, \quad \mathcal{C}_{RM} = [384, 8, 192]_2, \\
&\mathcal{C} = \mathcal{C}_{RS} \circ \mathcal{C}_{RM} = [17669, 128, 5952]_2, \quad (n_1 n_2 = 17664), \\
&\omega = 66, \quad \omega_r = \omega_e = 75
\end{aligned}$$

Table 6.3 below shows the security level estimates for the new code.

	Category I ($2n = 35338$)	Category III ($2n = 71702$)	Category V ($2n = 115274$)
Prange	166	237	300
Stern	145	213	276
Both-May (BM)	146	214	276

Table 6.3: The security level estimates computed using the syndrome decoding estimator [BE21] for the concatenated code of a shortened RS code and a duplicated RM code [EB21b, p. 25].

When designing new codes in the following Section, the goal is to match or improve the security level estimates in Table 6.3 without increasing the size of the public key which is $2n$ due to the index $s = 2$ of the quasi-cyclic code.

6.1 A New Choice of $\mathcal{C}$

Preferrably, the code $\mathcal{C}$ should satisfy each of the following:

- HQC should be secure: The security level estimates should be at least as big as the security level estimates in Table 6.3.
- A public key size which is not larger than the existing size from the concatenated code of a shortened RS code and a duplicated RM code.

- High error decoding capability.
- Small decoding failure rate.
- Efficient decoding in terms of computational speed.

With a large n, it is more likely that a good security level can be achieved, but on the other hand we want n to be small to reduce the key size. Hence, it is a tradeoff between good enough security level and small enough key sizes.

In this project, the following procedure has been used in order to design new codes:

1. Choose whether a product code or a concatenated code is used and choose code classes.
2. Fix the dimension k_1k_2. Note that the dimension depends on the specific security category.
3. Fix the length n_1n_2: Not too large as the public key size should not increase.
4. Derive the minimum distances d_1 and d_2.
5. Compute the security level estimates using the syndrome decoding estimator in [BE21]. If they are acceptable, try it out. Otherwise, try to optimize the parameters.

6.2 Code Design and Simulations

In this Section, the new codes that have been designed will be presented. Furthermore, the simulation experiments will be explained, and the results will be presented. Afterwards, the results will be discussed.

In one simulation, $10,000$ trials have been run, and the following have been measured:

For given $n_1, n_2, n, k, \delta, \omega, \omega_r, \omega_e$,

- In how many of the $10,000$ trials was it possible to derive the correct message $\mathbf{m}$ from decryption using the secret key sk? I.e. in how many trials was it possible to decode $\mathbf{v}'$ correctly using $\mathcal{C}$.Decode($\mathbf{v}'$)?
- What is the minimum and maximum values of the weight of the error that makes the codeword $\mathbf{v}'$ erroneous across all $10,000$ trials? This means the minimum and maximum values of $wt(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$.

- In how many of the $10,000$ trials was it possible to successfully decode $\mathbf{v}$, i.e. the second half of the ciphertext $ct = (\mathbf{u}, \mathbf{v})$? The goal is to get $0/10,000$ as decoding $\mathbf{v}$ means that the message can be decrypted without the secret key indicating that the encryption scheme is not secure. Note that it this thesis, a parameter ω_h has been invented. The parameter shows the weight of $\mathbf{h}$ in trials where it was possible to decode $\mathbf{v}$.
- What is the minimum and maximum values of the weight of the error that makes the codeword $\mathbf{v}$ erroneous across all $10,000$ trials? This means the minimum and maximum values of $wt(\mathbf{sr}_2 + \mathbf{e}')$. Preferrably, this weight should be such that $\mathbf{v}$ cannot be decoded.
- The security level estimates computed using the syndrome decoding estimator in [BE21].

To summarize the key information measured is how often it is possible to decode $\mathbf{v}'$ as this indicates the success rate of the encryption scheme, as well as how often $\mathbf{v}$ can be decoded, as this indicates the vulnerability of the encryption scheme.

Each simulation consists of 10,000 trials unless otherwise stated. One trial consists of running the three HQC algorithms, key generation, encryption, and decryption. As such each trial consists of 4 steps:

1. A pair of keys (public pk key and secret key sk) is generated.
2. A random message $\mathbf{m}$ of length $k_1 k_2$ is generated.
3. $\mathbf{m}$ is encrypted to a ciphertext $\mathbf{c}$ using the public key pk
4. $\mathbf{c}$ is decrypted using the secret key sk.

The simulations were run on DTU DCC's HPC cluster hardware [DTU21], which is a Linux machine with 24 intel(r) xeon(r) cpu e5-2660 v3 @ 2.60ghz cores and 256 GB of RAM.

6.2.1 Concatenated Codes

In this Section, the new concatenated codes that have been designed will be presented. Table 6.4 shows an overview of the concatenated codes. Note that the first code in Table 6.4, "Concat 1", is the original HQC concatenated code from the submission to the standardization process. "Concat 2" is a concatenated code of an RS code and an RM code, but without shortening the RS code or duplicating the RM code. The comparison of "Concat 1" and "Concat 2" provides an indication of what can be

achieved by shortening and duplicating. "Concat 3" is similar to "Concat 2" in terms of parameters, but the RM code has been replaced with a BCH code.

Due to the construction of a concatenated code, the first code $\mathcal{C}_1$ should be in a larger field than the second code $\mathcal{C}_2$. As the only non-binary code that has been implemented is the RS code, $\mathcal{C}_1$ was restricted to the RS code class. Among the implemented error-correcting codes, only the RM and BCH codes were possible classes for $\mathcal{C}_2$ as neither the repetition code with $k = 1$ nor the RS code with the bound $n \leq q - 1$ were suitable as $\mathcal{C}_2$ in the concatenated code construction.

Table 6.4 shows the chosen parameters, with alternative parameters being discussed in the following Sections. Table 6.4 also shows which security category each of the concatenated codes aims to match.

Recall that δ in the last column of Table 6.4 is the number of errors we expect the code to at least be able to correct. It is calculated as $\delta = \lfloor \frac{d_1-1}{2} \rfloor \lfloor \frac{d_2-1}{2} \rfloor$.

Reference	Category	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C} = \mathcal{C}_1 \circ \mathcal{C}_2$	δ
Concat 1	I	Shortened RS $[46, 16, 31]_{2^8}$	Duplicated RM $[384, 8, 182]_2$	$[17669, 128, \geq 5952]_2$ $(n_1 n_2 = 17664)$	≥ 1350
Concat 2	I	RS $[138, 16, 123]_{2^8}$	RM $[128, 8, 64]_2$	$[17669, 128, \geq 7872]_2$ $(n_1 n_2 = 17664)$	≥ 1891
Concat 3	I	RS $[139, 16, 124]_{2^8}$	BCH $[127, 8, \geq 61]_2$	$[17669, 128, \geq 7564]_2$ $(n_1 n_2 = 17653)$	≥ 1830

Table 6.4: Designed concatenated codes.

In the following Sections, the simulation results are presented.

6.2.1.1 Concat 1: Shortened RS Code and Duplicated RM Code

This is the security category I version of the original HQC concatenated code from the submission to the standardization process. Table 6.5 shows the simulation results. Note that the simulations in Table 6.5 have been run using the `C` code from the submission [Agu+21b].

Table 6.5 shows that for $\omega = 66$ and $\omega_r = \omega_e = 75$ (the values from the submission), it is, as expected, possible to successfully decode $\mathbf{v}'$ in $10,000/10,000$ trials. I.e. in every trial, it was possible to retrieve the correct message $\mathbf{m}$ from the Decryption Algorithm 5 of the cryptosystem. For these values of ω, ω_r, and ω_e, the minimum and maximum values of the weight of the error that makes $\mathbf{v}'$ an erroneous codeword of $\mathcal{C}$, i.e. the minimum and maximum values of $wt(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$, are 6,087 and 9,109, respectively. Hence, it is possible to correct more errors than the lower bound $\delta \geq 1350$ from Table 6.4. It was not possible to decode $\mathbf{v}$ in any of the $10,000$ trials. It is interesting to note the overlap between the weights of the error that makes $\mathbf{v}'$

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{s}\mathbf{r}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 66$ $\omega_r = 75$ $\omega_e = 75$	$10,000/$ $10,000$	$[6,087\ ;\ 9,109]$	$[8,551\ ;\ 9,081]$	$0/10,000$	Prange: 166 Stern: 145 BM: 146
$\omega = 80$ $\omega_r = 80$ $\omega_e = 80$	$10,000/$ $10,000$	$[6,760\ ;\ 9,080]$	$[8,548\ ;\ 9,074]$	$0/10,000$	Prange: 201.6 Stern: 179.4 BM: 179.9
$\omega = 85$ $\omega_r = 85$ $\omega_e = 85$	$9,919/$ $10,000$	$[7,113\ ;\ 9,087]$	$[8,557\ ;\ 9,107]$	$0/10,000$	Prange: 211.6 Stern: 189.1 BM: 189.6
$\omega = 90$ $\omega_r = 90$ $\omega_e = 90$	$1,451/$ $10,000$	$[7,458\ ;\ 9,096]$	$[8,582\ ;\ 9,082]$	$0/10,000$	Prange: 221.7 Stern: 198.9 BM: 199.3

Table 6.5: Simulation with 10,000 trials with the `C` code. Concatenated code with parameters $[17669, 128, \geq 5952]_2$ ($n_1 n_2 = 17664$) of a shortened RS code with parameters $[46, 16, 31]_{2^8}$ and a duplicated RM code with parameters $[384, 8, 182]_2$.

erroneous (column 3) and the weights of the error that makes $\mathbf{v}$ erroneous (column 4). This is discussed in Section 6.3.

Table 6.5 shows that even with an increase of the weight parameters to $\omega = \omega_r = \omega_e = 80$, it is still possible to obtain $10,000/10,000$ successful decodings of $\mathbf{v}'$ and $0/10,000$ decodings of $\mathbf{v}$. When the parameters ω, ω_r, and ω_e are increased further, then the number of successful decodings of $\mathbf{v}'$ decreases as expected.

Note that the security level estimates for $\omega = 66$ and $\omega_r = \omega_e = 75$ are taken from [EB21b, p. 25] and are the same as the estimates in the category I column in Table 6.3.

6.2.1.2 Concat 2: RS Code and RM Code

The parameters for the RM code were chosen to be the original parameters for the RM code used in "Concat 1" before the code was duplicated. As such, the parameters of the RM code are $[128, 8, 64]_2$ [Agu+21a]. The parameters for the RS code are then derived such that the parameters of the derived concatenated code match the parameters of "Concat 1" to allow for comparison. Table 6.6 shows the results of various simulations with 10,000 trials. These simulations have been run using the SageMath implementation presented in Chapter 5.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{s}\mathbf{r}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimate
$\omega = 61$ $\omega_r = 61$ $\omega_e = 61$	$9,994/$ $10,000$	$[4,879\ ;\ 5,253]$	$[6,422\ ;\ 12,589]$	$0/10,000$	Prange: 163.3 Stern: 142.8 BM: 143.3
$\omega = 62$ $\omega_r = 62$ $\omega_e = 62$	$9,781/$ $10,000$	$[4,998\ ;\ 5,394]$	$[3,192\ ;\ 9,086]$	$3/10,000$ $\omega_h = \{0, 1,$ $17663\}$	Prange: 165.3 Stern: 144.7 BM: 145.2
$\omega = 63$ $\omega_r = 63$ $\omega_e = 63$	$8,093/$ $10,000$	$[5,119\ ;\ 5,469]$	$[7,948\ ;\ 10,403]$	$0/10,000$	Prange: 167.4 Stern: 146.6 BM: 147.1
$\omega = 64$ $\omega_r = 64$ $\omega_e = 64$	$4,021/$ $10,000$	$[5,216\ ;\ 5,586]$	$[7,362\ ;\ 9,094]$	$0/10,000$	Prange: 169.4 Stern: 148.5 BM: 149.0

Table 6.6: Simulation with 10,000 trials with the SageMath code. Concatenated code with parameters $[17669, 128, \geq 7872]_2$ ($n_1 n_2 = 17664$) of an RS code with parameters $[138, 16, 123]_{2^8}$ and an RM code with parameters $[128, 8, 64]_2$.

Table 6.6 shows that with $\omega = \omega_r = \omega_e = 61$, it is possible to achieve $9,994/10,000$ successful decodings of $\mathbf{v}'$ with security level estimates that approximately match the category I security levels in Table 6.3. In order to achieve $10,000/10,000$ successful decodings of $\mathbf{v}'$, one or some of the parameters ω, ω_r, and ω_e should be lowered to 60, and then the security level estimates would probably be a bit below the preferred values in Table 6.3.

Table 6.6 shows that for $\omega = \omega_r = \omega_e = 62$, it was possible to decode $\mathbf{v}$ in $3/10,000$ trials. Recall from Section 6.2 that the parameter ω_h shows the weight of $\mathbf{h}$ in trials where it was possible to decode $\mathbf{v}$. Hence, Table 6.6 shows that in the 3 trials where it was possible to decode $\mathbf{v}$, the weight of $\mathbf{h}$ was 0, 1, and $17663 = n_1 n_2 - 1$, respectively. This will be further investigated in Section 6.2.4.

In general, it is interesting whether another set of parameters would yield better results. However, when testing different parameters, it is important not to significantly increase the size of the public key. Therefore, if the length of $\mathcal{C}_1$ is increased, then the length of $\mathcal{C}_2$ should be decreased, or vice versa.

An example where the length of $\mathcal{C}_1$ is decreased is the following concatenated code: Let $\mathcal{C}_1$ be an RS code with parameters $[69, 15, 55]_{2^9}$, and let $\mathcal{C}_2$ be an RM code with parameters $[256, 9, 128]_2$, then the concatenated code $\mathcal{C}$ has parameters $[17669, 135, \geq 7040]_2$ ($n_1 n_2 = 17664$) and $\lfloor \frac{d_1 - 1}{2} \rfloor \lfloor \frac{d_2 - 1}{2} \rfloor \geq 1701$. As the lower bound of δ for this code is

lower than the lower bound of δ for "Concat 2", this alternative set of parameters is not expected to be better.

An example where the length of $\mathcal{C}_2$ is decreased is the following concatenated code: Let $\mathcal{C}_2$ be an RM code with parameters $[64, 7, 32]_2$, then the RS code must be in the extension field $\mathbb{F}_{2^7}$ due to the dimension of the RM code and the construction of a concatenated code. This would imply that $n_{RS} \leq q - 1 = 127$ and hence the concatenated code becomes too short. In conclusion, with this quick analysis no better set of parameters for "Concat 2" was found.

It is interesting to notice that (the lower bound of) δ of "Concat 2" was actually greater than (the lower bound of) δ of "Concat 1". Hence, it would make sense to expect "Concat 2" to perform better than "Concat 1", but the simulations showed the opposite. This will be discussed in Section 6.3.

6.2.1.3 Concat 3: RS Code and BCH Code

In this Section, the simulation results using "Concat 3" from Table 6.4 are presented. "Concat 3" has similar parameters to "Concat 2", but the RM code has been replaced by a BCH code. The RM code from "Concat 2" is a first-order RM code (see Section 3.7), and in general first-order RM codes have really good minimum distance. On the other hand, the exact minimum distance of a BCH code is not easy to derive (see Section 3.5), so perhaps "Concat 3" could surprise. Table 6.7 shows the simulation results.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{s}\mathbf{r}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimate
$\omega = 51$ $\omega_r = 51$ $\omega_e = 51$	$9,988/$ $10,000$	$[3,811\ ;\ 4,125]$	$[2,286\ ;\ 12,433]$	$2/10,000$ $\omega_h = \{0, 1\}$	Prange: 143.2 Stern: 123.8 BM: 124.2
$\omega = 52$ $\omega_r = 52$ $\omega_e = 52$	$8,856/10,000$ $10,000$	$[3,954\ ;\ 4,232]$	$[4,122\ ;\ 9,102]$	$1/10,000$ $\omega_h = 17652$	Prange: 145.2 Stern: 125.7 BM: 126.1
$\omega = 53$ $\omega_r = 53$ $\omega_e = 53$	$2,346/10,000$ $10,000$	$[4,051\ ;\ 4,351]$	$[4,161\ ;\ 13,398]$	$0/10,000$	Prange: 147.3 Stern: 127.6 BM: 128.0

Table 6.7: Simulation with 10,000 trials with the SageMath code. Concatenated code with parameters $[17669, 128, \geq 7564]_2$ ($n_1 n_2 = 17653$) of an RS code with parameters $[139, 16, 124]_{2^8}$ and a BCH code with parameters $[127, 8, \geq 61]_2$.

Table 6.7 shows that for $\omega = \omega_r = \omega_e = 51$, it was possible to achieve $9,988/10,000$

successful decodings of $\mathbf{v}'$, and for these parameters, the security level estimates are lower compared to what could be obtained with "Concat 1" and "Concat 2". Furthermore, in order to achieve $10,000/10,000$ successful decodings of $\mathbf{v}'$, the value of one or more of ω, ω_r, or ω_e should be reduced to 50 which would result in an even lower security level estimate.

It was possible to decode $\mathbf{v}$ in 3 trials in total in Table 6.7 with $\omega_h = \{0, 1, 17652 = n_1 n_2 - 1\}$. This is exactly the same pattern as for "Concat 3", and this will be further investigated in Section 6.2.4.

It will briefly be considered if better results could be achieved by an alternative set of parameters: Again, either the length of $\mathcal{C}_1$ should be increased while the length of $\mathcal{C}_2$ is decreased, or the other way around. An example where the length of $\mathcal{C}_1$ is increased is the following: Let $\mathcal{C}_1$ be an RS code with parameters $[167, 16, 152]_{2^8}$ and let $\mathcal{C}_2$ be a shortened BCH code with parameters $[106, 8, \geq 40]_2$. Then the concatenated code has parameters $[17702, 128, \geq 6080]_2$ and $\delta = \lfloor \frac{d_1-1}{2} \rfloor \lfloor \frac{d_2-1}{2} \rfloor \geq 1425$, hence the lower bound of this δ is lower than the lower bound of δ of "Concat 3".

An example where the length of $\mathcal{C}_2$ is increased is the following: Let $\mathcal{C}_1$ be an RS code with parameters $[70, 16, 55]_{2^8}$ and let $\mathcal{C}_2$ be a shortened BCH code with parameters $[254, 8, \geq 127]_2$. Then the concatenated code has parameters $[17780, 128, \geq 6985]_2$ and $\delta = \lfloor \frac{d_1-1}{2} \rfloor \lfloor \frac{d_2-1}{2} \rfloor \geq 1701$. The lower bound of this δ is also lower than the lower bound of δ of "Concat 3". Hence, with this quick analysis it was not possible to find better parameters for "Concat 3".

6.2.2 Product Codes

In this Section, the new product codes that have been designed will be presented. Table 6.8 shows an overview of the product codes. Note that the first code in Table 6.8, "Product 1" is the original HQC product code used in the first version of HQC submitted to NIST in 2017. Recall that a product code is constructed using two codes over the same field. As three different binary codes have been implemented (RM, BCH, and repetition), there are a lot of possibilities. A product code of a shortened BCH code and an RM code instead of the repetition code will be investigated, as well as a product code of a shortened BCH code and another BCH code. Furthermore, two different product codes of two different RM codes will be investigated, and finally a product code of two RS codes over the extension field $\mathbb{F}_{2^8}$. Note that the parameters of this product code, i.e. "Product 5", look different compared to the other product codes. This is because it is the only non-binary product code that has been designed.

Note that all the new product codes have been designed such that their lengths match the length of "Concat 1" as "Concat 1" is the currently best code for HQC.

Reference	Category	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C} = \mathcal{C}_1 \circ \mathcal{C}_2$	δ
Product 1	I	Shortened BCH $[766, 256, 115]_2$	Repetition $[31, 1, 31]_2$	$[23869, 256, \geq 3751]_2$ $(n_1 n_2 = 23746)$	≥ 855
Product 2	I	Shortened BCH $[138, 22, \geq 31]_2$	RM $[128, 8, 64]_2$	$[17669, 176, \geq 1984]_2$ $(n_1 n_2 = 17664)$	≥ 465
Product 3	I	Shortened BCH $[139, 23, \geq 31]_2$	BCH $[127, 8, \geq 57]_2$	$[17669, 184, \geq 1767]_2$ $(n_1 n_2 = 17653)$	≥ 420
Product 4	I	RM $[512, 256, 32]_2$	Repetition $[33, 1, 33]_2$	$[16901, 256, 1056]_2$ $(n_1 n_2 = 16896)$	240
Product 5	I	RS $[47, 4, 44]_{2^8}$	RS $[47, 4, 44]_{2^8}$	$[17747, 128, \geq 1936]_{2^8}$ $(n_1 n_2 = 17672)$	$441 \leq \delta \leq 441 \cdot 8 = 3528$
Product 6	I	RM $[64, 7, 32]_2$	RM $[256, 37, 64]_2$	$[16421, 259, 2048]_2$ $(n_1 n_2 = 16384)$	465
Product 7	III	RM $[128, 29, 32]_2$	RM $[256, 9, 128]_2$	$[32771, 261, 4096]_2$ $(n_1 n_2 = 32768)$	945

Table 6.8: Product codes.

6.2.2.1 Product 1: Shortened BCH Code and Repetition Code

This is the category I parameters of the product code that was used in the first submission of HQC to the standardization process [Agu+21b]. Table 6.9 shows the simulation results with the `C` implementation from the submission [Agu+21b], and Table 6.10 shows the same simulations but with the SageMath implementation. By including simulations with both implementations, it is possible to compare the SageMath implementation described in Chapter 5 with the original `C` code.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{s}\mathbf{r}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 67$ $\omega_r = 77$ $\omega_e = 77$	$10,000/$ $10,000$	$[6,741\ ;\ 7,165]$	$[11,636\ ;\ 12,229]$	$0/10,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 77$ $\omega_r = 77$ $\omega_e = 77$	$10,000/$ $10,000$	$[7,363\ ;\ 7,801]$	$[11,661\ ;\ 12,220]$	$0/10,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 78$ $\omega_r = 78$ $\omega_e = 78$	$10,000/$ $10,000$	$[7,492\ ;\ 7,916]$	$[11,594\ ;\ 12,238]$	$0/10,000$	Prange: 198.6 Stern: 176.7 BM: 177.2

Table 6.9: Simulation with 10,000 trials with the C code. Product code with parameters $[23869, 256, \geq 3751]_2$ $(n_1 n_2 = 23746)$ of a shortened BCH code with parameters $[766, 256, 115]_2$ and a repetition code with parameters $[31, 1, 31]_2$.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{sr}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 67$ $\omega_r = 77$ $\omega_e = 77$	$10,000/$ $10,000$	$[6,743\ ;\ 7,157]$	$[11,600\ ;\ 16,803]$	$0/10,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 77$ $\omega_r = 77$ $\omega_e = 77$	$10,000/$ $10,000$	$[7,339\ ;\ 7,781]$	$[9,290\ ;\ 12,936]$	$0/10,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 78$ $\omega_r = 78$ $\omega_e = 78$	$10,000/$ $10,000$	$[7,422\ ;\ 7,886]$	$[4,862\ ;\ 12,206]$	$1/10,000$ $\omega_h = 0$	Prange: 198.6 Stern: 176.7 BM: 177.2

Table 6.10: Simulation with 10,000 trials with the SageMath code. Same product code as in Table 6.9.

Table 6.9 and 6.10 show that the SageMath implementation indeed is comparable to the `C` implementation. For all simulations in both Tables, it was possible to successfully decode $\mathbf{v}'$ in $10,000/10,000$ trials. Furthermore, the values in column 3, i.e. the minimum and maximum of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$, are almost identical in the two different Tables. However, the values in column 4, i.e. the minimum and maximum of $wt(\mathbf{sr}_2 + \mathbf{e}')$, are far from being identical. This is discussed in Sections 6.2.4 and 6.3.

Note that the security level estimates match those in Table 6.2.

6.2.2.2 Product 2: Shortened BCH Code and RM Code

In this product code, the repetition code has been replaced with an RM code. Furthermore, the parameters have been chosen such that n matches the n from "Concat 1" as "Concat 1" is the currently best choice of $\mathcal{C}$ for HQC [Agu+21a, p. 3].

The security level estimates for this code are too low compared to the values in the Tables 6.2 and 6.3. For $\omega = \omega_r = \omega_e = 55$ it was also only possible to achieve $9,956/10,000$ successful decodings of $\mathbf{v}'$, so one or some of the parameters ω, ω_r, and ω_e should be increased to 54 to achieve $10,000/10,000$ successful decodings of $\mathbf{v}'$. Hence, "Product 1" is indeed a better choice of $\mathcal{C}$ than "Product 2" which was as expected as a repetition code has an amazing minimum distance.

Table 6.11 shows that it was possible to decode $\mathbf{v}$ in 1 trial for $\omega_h = 1$. This is further discussed in Sections 6.2.4 and 6.3.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{s}\mathbf{r}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 55$ $\omega_r = 55$ $\omega_e = 55$	$9,956/$ $10,000$	$[4,255\ ;\ 4,581]$	$[5,700\ ;\ 12,028]$	$0/10,000$	Prange: 151.3 Stern: 131.4 BM: 131.8
$\omega = 56$ $\omega_r = 56$ $\omega_e = 56$	$9,660/$ $10,000$	$[4,392\ ;\ 4,710]$	$[4,578\ ;\ 9,120]$	$1/10,000$ $\omega_h = 1$	Prange: 153.3 Stern: 133.3 BM: 133.7

Table 6.11: Simulation with 10,000 trials. Product code with parameters $[17669, 176, \geq 1984]_2$ ($n_1 n_2 = 17664$) of a shortened BCH code with parameters $[138, 22, \geq 31]_2$ and an RM code with parameters $[128, 8, 64]_2$.

6.2.2.3 Product 3: Shortened BCH Code and BCH Code

This product code is very similar to "Product 2", but the RM code has been replaced with a BCH code. Again, the length of the product code has been chosen such that it matches the length of "Concat 1". The simulation results are shown in Table 6.12.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{s}\mathbf{r}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 43$ $\omega_r = 43$ $\omega_e = 43$	$10,000/$ $10,000$	$[2,943\ ;\ 3,167]$	$[1,712\ ;\ 14,626]$	$1/10,000$ $\omega_h = 0$	Prange: 127.2 Stern: 108.7 BM: 109.2
$\omega = 44$ $\omega_r = 44$ $\omega_e = 44$	$9,946/$ $10,000$	$[3,048\ ;\ 3,296]$	$[1,772\ ;\ 9,090]$	$4/10,000$ $\omega_h = 0, 0,$ $1, 17652$	Prange: 129.2 Stern: 110.6 BM: 111.0

Table 6.12: Simulation with 10,000 trials. A product code with parameters $[17669, 184, \geq 1767]_2$ ($n_1 n_2 = 17653$) of a shortened BCH code with parameters $[139, 23, \geq 31]_2$ and a BCH code with parameters $[127, 8, \geq 57]_2$.

Table 6.12 shows that the security level estimates for this code are even lower than the security level estimates for "Product 2" which makes sense as the first-order RM code has a good minimum distance.

Table 6.12 shows that in 5 trials in total it was possible to decode $\mathbf{v}$. In these 5 trials, the weight of $\mathbf{h}$ was $0, 0, 0, 1, 17652 = n_1 n_2 - 1$, respectively. See Section 6.2.4 for a further investigation in this.

6.2.2.4 Product 4: RM Code and Repetition Code

In this product code, the shortened BCH code has been replaced with a fourth-order RM code which has been combined with a repetition code. Table 6.13 shows the simulation results.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{x}\mathbf{r}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{s}\mathbf{r}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 62$ $\omega_r = 62$ $\omega_e = 62$	$9,996/$ $10,000$	$[4,904\ ;\ 5,276]$	$[5,044\ ;\ 8,758]$	$3/10,000$ $\omega_h = 16895,$ $16895, 16895$	Prange: 165.2 Stern: 144.5 BM: 145.0

Table 6.13: Simulation with 10,000 trials. A product code with parameters $[16901, 256, 1056]_2$ ($n_1 n_2 = 16896$) of an $\mathcal{RM}(4, 9)$ code with parameters $[512, 256, 32]_2$ and a repetition code with parameters $[33, 1, 33]_2$.

Table 6.13 shows that the product code achieves very good security level estimates of 144.5 to 165.2 matching those of "Concat 1" and simultaneously allowing for a key size of 16901 vs. 17669 from "Concat 1". This is surprising given that "Product 4" has the lowest δ of all the designed product codes according to Table 6.8 and 6.2.

For $\omega = \omega_r = \omega_e = 62$, $9,996/10,000$ successful decodings of $\mathbf{v}'$ was achieved. In order to achieve $10,000/10,000$ successful decodings of $\mathbf{v}'$, one or more of the parameters ω, ω_r, and ω_e should be lowered to 61.

In $3/10,000$ trials, it was possible to decode $\mathbf{v}$, and in all 3 trials the weight of $\mathbf{h}$ was $16895 = n_1 n_2 - 1$. The fact that the weight of $\mathbf{h}$ becomes exactly this value in $3/10,000$ cases shows lack of randomness in the random number generation in the Sagemath implementation.

6.2.2.5 Product 5: RS Code and RS Code

This product code is different from the rest in the sense that it is over the field $\mathbb{F}_{2^8}$ and not over the binary field $\mathbb{F}_2$. Therefore, the parameters are also intepreted a bit differently: Two codes, each of length $n = 47$ sounds quite short. The reason for such short codes is that the length $n = 47 \cdot 47 = 2209$ is in the extension field $\mathbb{F}_{2^8}$ which means that the corresponding length in the base field $\mathbb{F}_2$ (which is also the field HQC works in) is $8 \cdot 2209 = 17672$. The same holds for the dimension: The dimension of the product code in the extension field is $4 \cdot 4 = 16$ which means that it is $8 \cdot 16 = 128$ in the base field. Unfortunately, the same does not completely hold for the minimum distance. The minimum distance could be multiplied with 8 in the same manner if and only if only blocks of length 8 of errors, i.e. 8 consecutive errors, occurred (in coding theory, such blocks of errors are called *burst errors*), and also always took place

in the right places in the bit stream, i.e. that the blocks of errors never overlap the blocks of 8 bits. Therefore, we only know that the minimum distance is somewhere between $44 \cdot 44 = 1936$ and $8 \cdot 1936 = 15,488$ and probably much closer to 1936 than to 15,488. The simulation results are shown in Table 6.14.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{sr}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 24$ $\omega_r = 24$ $\omega_e = 24$	$10,000/$ $10,000$	$[1,054\ ;\ 1,144]$	$[1,596\ ;\ 9,066]$	$0/10,000$	Prange: 89.1 Stern: 74.1 BM: 74.5
$\omega = 25$ $\omega_r = 25$ $\omega_e = 25$	$9,854/$ $10,000$	$[1,141\ ;\ 1,235]$	$[2,203\ ;\ 16,493]$	$0/10,000$	Prange: 91.1 Stern: 75.8 BM: 76.3

Table 6.14: Simulation with 10,000 trials. Product code with parameters $[17747, 128, \geq 1936]_{2^8}$ ($n_1 n_2 = 17672$) of two RS codes, both with parameters $[47, 4, 44]_{2^8}$.

Table 6.14 shows that the security level estimates that could be achieved by using this code are very low. They are the lowest seen so far.

6.2.2.6 Product 6: RM Code and RM Code

This product code is constructed using two different RM codes, a first-order $\mathcal{RM}(1, 6)$ code and a second-order $\mathcal{RM}(2, 8)$ code. Due to the restrictions of possible parameter sets for RM codes, it was not easy to find some very suitable parameters. The length n is a bit smaller than the rest of the designed codes while the dimension k is quite big for a security category I encryption scheme to be.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{sr}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 39$ $\omega_r = 39$ $\omega_e = 39$	$9,967/$ $10,000$	$[2,479\ ;\ 2,683]$	$[3,464\ ;\ 13,797]$	$0/10,000$	Prange: 118.9 Stern: 101.0 BM: 101.4
$\omega = 40$ $\omega_r = 40$ $\omega_e = 40$	$9,749/$ $10,000$	$[2,574\ ;\ 2,786]$	$[3,672\ ;\ 8,440]$	$0/10,000$	Prange: 120.9 Stern: 102.8 BM: 103.3

Table 6.15: Simulation with 10,000 trials. Product code with parameters $[16421, 259, 2048]_2$ ($n_1 n_2 = 16384$) of an $\mathcal{RM}(1, 6)$ code with parameters $[64, 7, 32]_2$ and an $\mathcal{RM}(2, 8)$ with parameters $[256, 37, 64]_2$.

Again, the security level estimates that would be achieved are too low compared to Table 6.3.

6.2.2.7 Product 7: RM Code and RM Code

This product code is also constructed using two different RM codes, an $\mathcal{RM}(2,7)$ code and an $\mathcal{RM}(1,8)$ code, and the aim of this code is to match the security category III. As before, it was challenging to find a suitable set of parameters for a product code of two different RM codes.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{sr}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 83$ $\omega_r = 83$ $\omega_e = 83$	$9,988/$ $10,000$	$[9,149\ ;\ 9,725]$	$[13,433\ ;\ 23,389]$	$0/10,000$	Prange: 209.9 Stern: 187.6 BM: 188.1
$\omega = 84$ $\omega_r = 84$ $\omega_e = 84$	$9,942/$ $10,000$	$[9,288\ ;\ 9,782]$	$[11,970\ ;\ 16,732]$	$0/10,000$	Prange: 211.9 Stern: 189.6 BM: 190.0

Table 6.16: Simulation with 10,000 trials. Product code with parameters $[32771, 261, 4096]_2$ ($n_1 n_2 = 32768$) of an $\mathcal{RM}(2,7)$ code with parameters $[128, 29, 32]_2$ and an $\mathcal{RM}(1,8)$ code with parameters $[256, 9, 128]_2$.

As this product code aims to match security category III, the security level estimates in Table 6.16 should be compared to the category III column in Table 6.3. The estimate of Prange's algorithm is 209.9 in Table 6.16 while it is 237 in Table 6.3, hence the security level estimates in Table 6.16 are too low.

6.2.3 Single Codes

A natural question is why $\mathcal{C}$ is chosen as a combination of two codes instead of just as one single code. This will be investigated in this Section.

In general, an advantage of a concatenated/product code is that longer codes can be achieved while still being able to decode efficiently.

With the available code classes that are covered in this thesis, the single code can be either an RS, RM, BCH, or repetition code. A really large RS code would require an enormous field because of the bound $n \leq q - 1$. An enormous field would imply inefficient decoding as the computational complexity is increased, and therefore RS code is not a suitable choice. The same holds for the BCH code as the BCH code is decoded using the RS decoder. Furthermore, the repetition code is not suitable either as it has $k = 1$. Hence, an RM code seems like the most promising choice.

Two different single codes have been designed, one for security category I and one for security category III. The parameters are shown in Table 6.17.

Reference	Category	$\mathcal{C}$	δ
Single 1	I	RM $[16421, 470, 2048]_2$ $(n_1 = 16384)$	1023
Single 2	III	RM $[32771, 121, 8192]_2$ $(n_1 = 32768)$	4095

Table 6.17: Single codes.

However, note that the dimension of "Single 1" is quite big compared to the preferred value ≥ 128, while the dimension of "Single 2" is way too small compared to the preferred value ≥ 192 for security category III. This indicates that no suitable single code among the code classes covered in this thesis could be found.

Due to time restrictions, only 5,000 trials per simulation have been run in this Section.

6.2.3.1 Single 1: RM Code

This code is an $\mathcal{RM}(r = 3, m = 14)$ code. As mentioned, the dimension is quite big for a security category I code. A way to decrease the dimension while keeping the same length would be to choose an $\mathcal{RM}(r = 2, m = 14)$ code with parameters $[16421, 106, 4096]_2$, but the dimension of this code is too small, and therefore the already chosen parameters seem better. Table 6.18 shows the simulation results.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{sr}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 33$ $\omega_r = 33$ $\omega_e = 33$	$5,000/5,000$	$[1,869\ ;\ 2,011]$	$[3,371\ ;\ 11,829]$	$0/5,000$	Prange: 106.8 Stern: 89.9 BM: 90.4
$\omega = 34$ $\omega_r = 34$ $\omega_e = 34$	$4,997/5,000$	$[1,976\ ;\ 2,128]$	$[2,850\ ;\ 8,402]$	$0/5,000$	Prange: 108.8 Stern: 91.7 BM: 92.2
$\omega = 35$ $\omega_r = 35$ $\omega_e = 35$	$4,947/5,000$	$[2,067\ ;\ 2,249]$	$[4,302\ ;\ 11,533]$	$0/5,000$	Prange: 110.8 Stern: 93.6 BM: 94.0

Table 6.18: Simulation with 5,000 trials. A single $\mathcal{RM}(3, 14)$ code.

Table 6.18 shows that the security level estimates are much lower than the values in Table 6.3, and hence this code is not suitable. The only designed code that this code outperforms is the product code of two RS codes, "Product 4".

6.2.3.2 Single 2: RM Code

This code is an $\mathcal{RM}(r = 2, m = 15)$ code, and the aim is to match security category III. As mentioned, the dimension is too small for a security category III code, as it should preferrably be ≥ 192 to encapsulate an AES192-key. A way to increase the dimension while keeping the same length would be to choose an $\mathcal{RM}(r = 3, m = 15)$ code with parameters $[32771, 576, 4096]_2$, but the dimension of this code is really big. The simulation results are shown in Table 6.19.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{sr}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level estimates
$\omega = 80$ $\omega_r = 80$ $\omega_e = 80$	$4,968/5,000$	$[8,740\ ;\ 9,154]$	$[11,448\ ;\ 16,702]$	$0/5,000$	Prange: 203.9 Stern: 181.8 BM: 182.3
$\omega = 81$ $\omega_r = 81$ $\omega_e = 81$	$4,899/5,000$	$[8,867\ ;\ 9,351]$	$[13,053\ ;\ 16,781]$	$0/5,000$	Prange: 205.9 Stern: 183.8 BM: 184.2
$\omega = 82$ $\omega_r = 82$ $\omega_e = 82$	$4,709/5,000$	$[8,948\ ;\ 9,444]$	$[11,638\ ;\ 16,724]$	$0/5,000$	Prange: 207.9 Stern: 185.7 BM: 186.2

Table 6.19: Simulation with 5,000 trials. A single $\mathcal{RM}(2, 15)$ code.

The security level estimates in Table 6.19 are too low compared to the category III column in Table 6.3.

Hence, it was not possible to find a well performing single code for neither security category I nor security category III. However, the analysis of the performance of using a single code as $\mathcal{C}$ in HQC has definitely not been thoroughly investigated yet, and therefore it is an interesting topic for future work.

6.2.4 Capability of Decoding $\mathbf{v}$

Throughout this Chapter, many simulations have shown that it was possible to decode $\mathbf{v}$ in a few trials. If $\mathbf{v}$ can be decoded, than half of the ciphertext $ct = (\mathbf{u}, \mathbf{v})$ can be decoded without the secret key sk which means that the encryption scheme is vulnerable. However, in the two Tables 6.5 and 6.9 showing simulation results with the `C` code, it is not possible to decode $\mathbf{v}$ in any trial.

By further investigating this, it was revealed that the generation of $\mathbf{h}$ has been implemented differently compared to the generation of $\mathbf{h}$ in the `C` code as mentioned in the beginning of the Chapter. As a part of the further investigation, the following Table 6.20 was created.

Weight	min and max of ω_h
$\omega = 66$, $\omega_r = \omega_e = 75$	$[8,607\ ;\ 9,088]$

Table 6.20: An addition to the "Concat 1" Table 6.5 with simulation results computed with the `C` code. This Table shows the minimum and maximum weights of **h** denoted ω_h across all 10,000 trials. It can be seen that ω_h is always around $\lfloor \frac{n}{2} \rfloor = 8,834$.

Table 6.20 shows that across 10,000 trials with the `C` code, the minimum and maximum weight of **h** are 8,607 and 9,088, respectively. Hence, the weight of **h** is always close to $\lfloor \frac{n}{2} \rfloor = 8,834$. In the `C` code, every bit of **h** has the same probability of be a 1, and for a large n this means the $\omega_h \approx \frac{n}{2}$.

In the SageMath implementation, **h** is generated such that the weight of the vector can be any value between 0 and n, all with the same probability. This is of course unfortunate, but it can still be used to show new interesting aspects of HQC:

In Table 6.21, "Product 1" is used for new simulations. In these simulations, the weight of **h** has been fixed across all 5,000 trials per simulation. The different fixed weights of **h** are marked with red in Table 6.21.

In Table 6.21, ω_h has been fixed to values close to 0 and values close to $n_1 n_2$ as these values seem vulnerable according to all the simulations throughout the Chapter.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{sr}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level
$\omega = 67$ $\omega_r = 77$ $\omega_e = 77$ $\omega_h = 0$	$5,000/5,000$	$[6,747\ ;\ 7,123]$	$[4,128\ ;\ 4,400]$	$5,000/5,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 67$ $\omega_r = 77$ $\omega_e = 77$ $\omega_h = 1$	$5,000/5,000$	$[6,779\ ;\ 7,149]$	$[6,621\ ;\ 7,155]$	$5,000/5,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 67$ $\omega_r = 77$ $\omega_e = 77$ $\omega_h = 2$	$5,000/5,000$	$[6,763\ ;\ 7,195]$	$[8,382\ ;\ 8,990]$	$2,537/5,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 67$ $\omega_r = 77$ $\omega_e = 77$ $\omega_h = 3$	$5,000/5,000$	$[6,759\ ;\ 7,157]$	$[9,491\ ;\ 10,149]$	$0/5,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 67$ $\omega_r = 77$ $\omega_e = 77$ $\omega_h = n_1n_2-2$	$5,000/5,000$	$[6,765\ ;\ 7,165]$	$[14,784\ ;\ 15,352]$	$0/5,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 67$ $\omega_r = 77$ $\omega_e = 77$ $\omega_h = n_1n_2-1$	$5,000/5,000$	$[6,721\ ;\ 7,183]$	$[16,593\ ;\ 17,047]$	$0/5,000$	Prange: 196.6 Stern: 174.8 BM: 175.2
$\omega = 67$ $\omega_r = 77$ $\omega_e = 77$ $\omega_h = n_1n_2$	$5,000/5,000$	$[6,749\ ;\ 7,141]$	$[19,358\ ;\ 19,634]$	$0/5,000$	Prange: 196.6 Stern: 174.8 BM: 175.2

Table 6.21: Simulation with 5,000 trials. Product code with shortened BCH and Repetition code, code parameters $[766, 256, 121]_2$ and $[31, 1, 31]_2$ as in the submission. Note the $wt(h)$ has been added.

Table 6.21 shows that for this choice of $\mathcal{C}$, HQC is only vulnerable when ω_h is close to 0: For $\omega_h = 0$ and $\omega_h = 1$, it was possible to decode $\mathbf{v}$ in all $5,000/5,000$ trials. For $\omega_h = 2$, it was possible in approximately half of the trials. However, it was not possible to decode $\mathbf{v}$ for $\omega_h = \{n_1n_2 - 2, n_1n_2 - 1, n_1n_2\}$ for any trial.

Table 6.22 shows similar simulations for "Product 4", the best performing of the

designed codes.

Weight	$\mathcal{C}.Decode(\mathbf{v}') = \mathbf{m}$	min and max of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}'$)	min and max of $wt(\mathbf{sr}_2 + \mathbf{e}')$ (wt of error corrupting $\mathbf{v}$)	$\mathcal{C}.Decode(\mathbf{v}) = \mathbf{m}$	Security level
$\omega = 62$ $\omega_r = 62$ $\omega_e = 62$ $\omega_h = 0$	$4,999/5,000$	$[4,892\ ;\ 5,262]$	$[3,016\ ;\ 3,246]$	$5,000/5,000$	Prange: 165.2 Stern: 144.5 BM: 145.0
$\omega = 62$ $\omega_r = 62$ $\omega_e = 62$ $\omega_h = 1$	$4,999/5,000$	$[4,924\ ;\ 5,252]$	$[4,886\ ;\ 5,276]$	$4,999/5,000$	Prange: 165.2 Stern: 144.5 BM: 145.0
$\omega = 62$ $\omega_r = 62$ $\omega_e = 62$ $\omega_h = 2$	$4,999/5,000$	$[4,930\ ;\ 5,270]$	$[6,064\ ;\ 6,532]$	$0/5,000$	Prange: 165.2 Stern: 144.5 BM: 145.0
$\omega = 62$ $\omega_r = 62$ $\omega_e = 62$ $\omega_h = n_1n_2-2$	$4,999/5,000$	$[4,924\ ;\ 5,310]$	$[6,066\ ;\ 6,584]$	$0/5,000$	Prange: 165.2 Stern: 144.5 BM: 145.0
$\omega = 62$ $\omega_r = 62$ $\omega_e = 62$ $\omega_h = n_1n_2-1$	$5,000/5,000$	$[4,898\ ;\ 5,268]$	$[4,876\ ;\ 5,276]$	$5,000/5,000$	Prange: 165.2 Stern: 144.5 BM: 145.0
$\omega = 62$ $\omega_r = 62$ $\omega_e = 62$ $\omega_h = n_1n_2$	$4,999/5,000$	$[4,924\ ;\ 5,264]$	$[3,034\ ;\ 3,258]$	$5,000/5,000$	Prange: 165.2 Stern: 144.5 BM: 145.0

Table 6.22: Simulation with 5,000 trials. Product code with RM code and Repetition code. Note the $wt(h)$ has been added.

Table 6.22 shows that for this choice of $\mathcal{C}$ it is always possible to decode $\mathbf{v}$ for $\omega_h = \{0, 1, n_1n_2 - 1, n_1n_2\}$. Note that the values in column 4, i.e. the minimum and maximum values of the weight of the error corrupting $\mathbf{v}$, decrease in the last rows, while these values keep increasing in Table 6.21. This will be discussed in Section 6.3.

6.3 Discussion

First of all, it turned out that designing new codes is a difficult task as there are many details to take into account. Secondly, even though codes are carefully designed, they can still perform poorly with regards to the current purpose. Many of the codes that have been designed did not perform well as $\mathcal{C}$ in HQC. However, well performing choices of $\mathcal{C}$ were also seen.

The designed concatenated codes showed the following: "Concat 3", a concatenated code of an RS code and a BCH code, was outperformed by "Concat 2", a concatenated code of an RS code and an RM code. The two codes had approximately the same set of parameters, but in general first-order RM codes have very good minimum distance which was also expressed by the fact that δ for "Concat 2" was ≥ 1891 while it was ≥ 1830 for "Concat 3". Hence, theoretically it was expected that "Concat 2" would perform better than "Concat 3" which also turned out to be the case. Both "Concat 2" and "Concat 3" were outperformed by "Concat 1", the original HQC concatenated code of a shortened RS code and a duplicated RM code. Hence, it was not possible to find a better concatenated code than the one used in the submission to NIST.

The best performing among the designed codes was "Product 4", a product code of an $\mathcal{RM}(r = 4, m = 9)$ code and a repetition code of length $n = 33$. With this choice of $\mathcal{C}$, it was possible to achieve security level estimates that match the category I security level estimates of the current version of HQC. Additionally, "Product 4" allows for an even smaller public key size, as the public key size of the current HQC version is $2n = 2 \cdot 17669 = 35338$, while the public key size when using "Product 4" as $\mathcal{C}$ is $2n = 2 \cdot 16901 = 33802$. Hence, "Product 4" seems like a promising candidate.

However, "Product 4" still needs some further research and improvement as the parameters providing the security level estimates presented in Table 6.13 only resulted in $9,996/10,000$ successful decodings of $\mathbf{v}'$. The code can be improved in different ways. First of all, it should be investigated if $10,000/10,000$ successful decodings of $\mathbf{v}'$ can be achieved by decreasing some of the parameters ω, ω_r, and ω_e while keeping the security level estimates sufficient for security category I. Another way of improving the code could be to adjust the parameters, e.g. increase the length of the repetition code by a few positions.

It is interesting to notice that according to Tables 6.4 and 6.8, "Product 4" had the absolute lowest δ among all designed codes. It was a general pattern that it was difficult to predict the performance beforehand from the δ value, but of course it did not help that most of the δ values were given as a lower bound instead of an exact value. Table 6.4 showed that the lower bound of δ of e.g. "Concat 2" was much bigger than the lower bound of δ of "Concat 1", but "Concat 1" was still a better choice of $\mathcal{C}$. Hence, simulations are necessary in order to know how different choices of $\mathcal{C}$ perform in HQC.

The actual error correction capability can be interpreted from column 3 in all the tables. Consider e.g. the row with $\omega = 66$ and $\omega_r = \omega_e = 75$ in Table 6.5 from "Concat 1". It was possible to successfully decode $\mathbf{v}'$ in $10,000/10,000$ trials, and the minimum and maximum of $wt(\mathbf{xr}_2 - \mathbf{r}_1\mathbf{y} + \mathbf{e}')$ are $[6,087\ ;\ 9,109]$. This means that $\mathcal{C}$ can decode $9,109$ errors which is much greater than the $\delta \geq 1350$ derived from the parameters of the code.

Another interesting thing is the following: Again, consider the row with $\omega = 66$ and $\omega_r = \omega_e = 75$ in Table 6.5, and note the overlap between the values in column 3 and 4, i.e. the minimum value in column 3 is smaller than the minimum value in column 4, while the maximum value in column 3 is greater than the maximum value in column 4. This is interesting as $\mathbf{v}'$ could be decoded in $10,000/10,000$ trials, while $\mathbf{v}$ could be decoded in $0/10,000$ trials. This shows that the decoding capability depends on the distribution of errors: In the trial where the weight of the error corrupting $\mathbf{v}'$ was 9,109, the weight of the error corrupting $\mathbf{v}$ in the same trial must have been lower. At the same time, it was possible to decode $\mathbf{v}'$, but not $\mathbf{v}$ which proves the claim that the decoding capability depends on the distribution of errors.

The worst performing among the new codes was "Product 5", a product code over the field $\mathbb{F}_{2^8}$ of two identical RS codes. This indicates that binary product codes are potentially better than those designed using a larger field, but further research is necessary in order to finally conclude it.

As mentioned in Chapter 3, concatenated codes usually have better parameters than product codes in terms of minimum distance. Despite this the best performing code among the designed codes turned out to be a product code. This can potentially be explained by the fact that it was easier to design and test more varied product codes than concatenated codes. In order to design more concatenated codes, more non-binary codes are needed. Hence, for future research it is interesting to investigate other non-binary codes. It is also interesting to design codes of the improved version of the concatenated code called *generalized concatenated codes* [Wac+, p. 110].

In Section 6.2.3, the fact that $\mathcal{C}$ should be chosen as a product or concatenated code was challenged. Two different single codes were designed and simulated. However, the design of single codes as well as the simulations showed that no well performing single code could be found: The best possible set of parameters were not suitable for HQC, and the security level estimates that could be achieved were not acceptable either. However, this can still be researched more.

Finally, the comparison between the two Tables 6.9 and 6.10 allows for a comparison between the `C` code submitted to NIST and the SageMath code implemented in this project. The comparison shows that the SageMath code is comparable to the `C` code, particularly with regards to the decoding capability of $\mathbf{v}'$ and the weight of the error corrupting $\mathbf{v}'$. However, as already mentioned $\mathbf{h}$ is generated differently in the

SageMath implementation compared to the `C` code: A $\mathbf{h}$ generated with the `C` code has weight around $\frac{n}{2}$ with very high probability while a $\mathbf{h}$ generated with the SageMath implementation can have any weight between 0 and n with equal probability. In Section 6.2.4, it was shown that if the weight of $\mathbf{h}$ is either very close to 0 or very close to $n_1 n_2$, then $\mathbf{v}$ can be decoded with high probability (depending on the choice of $\mathcal{C}$), and hence the encryption scheme is not secure. In conclusion: It does not seem likely that $\mathbf{v}$ can be decoded when the `C` code is used, but if an attacker can compromise the encryption scheme and force $\mathbf{h}$ to have weight either very close to 0 or very close to $n_1 n_2$, then she can attack the encryption scheme, e.g. through fault attacks.

CHAPTER 7

Conclusion

First of all, it was possible to implement different error-correcting codes as well as product codes, concatenated codes, and the HQC encryption scheme in SageMath. The SageMath implementation was comparable to the `C` implementation from the submission to NIST's standardization process, particularly with regards to the decoding capability. Secondly, it was possible to design many new codes, both product codes, concatenated codes, and a couple of single codes, using different combinations of the error-correcting codes covered in this thesis.

Simulations with 10,000 trials per simulation were run on HPC cluster hardware. Of the designed codes, the best performing was a product code of an $\mathcal{RM}(r = 4, m = 9)$ code and a repetition code of length $n = 33$. These parameters were constructed to match security category I among the 5 categories that NIST has defined. With this choice of $\mathcal{C}$, it is possible to achieve security level estimates that match the estimates of the current best version of HQC where $\mathcal{C}$ is chosen as a concatenated code of a shortened Reed-Solomon code and a duplicated Reed-Muller code. Additionally, this new product code allows for an even smaller public key size: The concatenated code that is currently used in HQC has $n = 17669$ which means a public key size of $2n = 2 \cdot 17669 = 35338$ while this new product code has $n = 16901$ implying a public key size of $2n = 2 \cdot 16901 = 33802$. Hence, the new product code seems like a promising candidate for an improved choice of $\mathcal{C}$ in HQC.

Interestingly, the theoretical decoding capability of the new product code, i.e. the number of errors that the code was expected to at least be able to correct, did not seem promising. Theoretically, the code was expected to be among the worst performing of the designed codes. It was a general pattern throughout all the experiments that the theoretical decoding capability turned out to not be useful as a prediction for the simulation results. Hence, it can be concluded that simulations are necessary in order to conclude how different choices of $\mathcal{C}$ perform in HQC.

The simulations showed another interesting fact: If the n-bit vector $\mathbf{h}$ from the public key $pk = (\mathbf{h}, \mathbf{s})$ is chosen with Hamming weight very close to either 0 or n, then the n-bit vector $\mathbf{v}$ from the ciphertext $ct = (\mathbf{u}, \mathbf{v})$, can be decrypted without using the secret key sk which would mean that the encryption scheme was not secure. With the `C` implementation of HQC submitted to NIST's standardization process, it is really

unlikely, bordering impossible, that $\mathbf{h}$ could randomly get such a weight. However, if an attacker can compromise the encryption scheme and force the Hamming weight of $\mathbf{h}$ to be either very close to 0 or to n, then she can attack the encryption scheme, e.g. by fault attacks.

As HQC is a quite new cryptosystem and hence not yet thoroughly researched, it makes sense that the old Classic McEliece from 1978 performs better in the standardization process. However, Classic McEliece suffers from large key sizes, and on this parameter HQC performs much better. Perhaps, HQC can compete with Classic McEliece in the future.

7.1 Future Work

Some topics for future work are:

- Further research in the product code of an $\mathcal{RM}(r = 4, m = 9)$ code and a repetition code of length $n = 33$. Investigate whether this code is actually an improved choice of $\mathcal{C}$. Furthermore, design set of parameters for this product code for security category III and V as well.
- Implement and design codes of other classes of error-correcting codes, e.g. Goppa codes which is another subfield subcode of RS codes.
- Design generalized concatenated codes, an improved version of the concatenated code used in this thesis.
- Improve the implementation. Implement better decoders, e.g. list decoders. If the implementation is improved a lot, it would be interesting to contribute to the Sagemath community.
- Further investigation in single codes as $\mathcal{C}$.

Bibliography

[20221] PQCrypto 2021. *PQCrypto conference 2021: Live Session - Day 2*. July 2021. URL: https://www.youtube.com/watch?v=FdOKWktBLhU.

[Agu+21a] C. Aguilar Melchor et al. "Hamming Quasi-Cyclic (HQC), Updated version 06/06/2021". In: (2021). URL: https://pqc-hqc.org/doc/hqc-specification_2021-06-06.pdf.

[Agu+21b] C. Aguilar Melchor et al. "HQC Implementation". In: (2021). Last visited on 17th of January 2022. URL: https://pqc-hqc.org/implementation.html.

[BE21] Emanuele Bellini and Andre Esser. *Syndrome Decoding Estimator*. 2021. URL: https://github.com/Crypto-TII/syndrome_decoding_estimator.

[BM18] Leif Both and Alexander May. "Decoding Linear Codes with High Error Rate and Its Impact for LPN Security". en. In: *Post-Quantum Cryptography*. Edited by Tanja Lange and Rainer Steinwandt. Volume 10786. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2018, pages 25–46. ISBN: 978-3-319-79062-6 978-3-319-79063-3. DOI: 10.1007/978-3-319-79063-3_2. URL: http://link.springer.com/10.1007/978-3-319-79063-3_2.

[DTU21] DTU Computing Center. *DTU Computing Center resources*. 2021. DOI: 10.48714/DTU.HPC.0001. URL: https://doi.org/10.48714/DTU.HPC.0001.

[Dum96] I. Dumer. "Suboptimal decoding of linear codes: partition technique". In: *IEEE Transactions on Information Theory* 42.6 (November 1996). Conference Name: IEEE Transactions on Information Theory, pages 1971–1986. ISSN: 1557-9654. DOI: 10.1109/18.556688.

[EB21a] Andre Esser and Emanuele Bellini. *Syndrome Decoding Estimator*. Technical report 1243. 2021. URL: https://eprint.iacr.org/2021/1243.

[EB21b] Andre Esser and Emanuele Bellini. "Syndrome Decoding Estimator". In: *IACR Cryptol. ePrint Arch.* 2021 (2021), page 1243.

[IBMa] IBM. "Grover's algorithm". In: (). Last visited on 17th of January 2022. URL: https://quantum-computing.ibm.com/composer/docs/iqx/guide/grovers-algorithm.

[IBMb] IBM. “Shor’s algorithm”. In: (). Last visited on 17th of January 2022. URL: https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm.

[Knu18a] L.R. Knudsen. *Advanced Cryptology - How to Crack It 2*. 1st edition. Polyteknisk Forlag, 2018.

[Knu18b] L.R. Knudsen. *Cryptology - How to Crack It*. 1st edition. Polyteknisk Forlag, 2018.

[LB88] P. J. Lee and E. F. Brickell. “An Observation on the Security of McEliece’s Public-Key Cryptosystem”. en. In: *Advances in Cryptology — EUROCRYPT ’88*. Edited by D. Barstow et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1988, pages 275–280. ISBN: 978-3-540-45961-3. DOI: 10.1007/3-540-45961-8_25.

[MO15] Alexander May and Ilya Ozerov. “On Computing Nearest Neighbors with Applications to Decoding of Binary Linear Codes”. In: *EUROCRYPT (1)*. Springer, 2015, pages 203–228. DOI: 10.1007/978-3-662-46800-5_9. URL: https://www.iacr.org/archive/eurocrypt2015/90560136/90560136.pdf.

[MS77] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland Publishing Company, 1977.

[NIS21] NIST. “Post-Quantum Cryptography - Past Events”. In: (2021). Last visited on 17th of January 2022. URL: https://csrc.nist.gov/Projects/post-quantum-cryptography/events.

[Pra62] E. Prange. “The use of information sets in decoding cyclic codes”. In: *IRE Transactions on Information Theory* 8.5 (September 1962). Conference Name: IRE Transactions on Information Theory, pages 5–9. ISSN: 2168-2712. DOI: 10.1109/TIT.1962.1057777.

[RB20] J. Rosenkilde and P. Beelen. *Lecture notes for course 01405 Algebraic Error-Correcting Codes*. Technical University of Denmark, 2020.

[Rot06] R. Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006.

[Sch+20] T. Schamberger et al. “A Power Side-Channel Attack on the CCA2-Secure HQC KEM”. In: (July 2020).

[Ste89] Jacques Stern. “A method for finding codewords of small weight”. en. In: *Coding Theory and Applications*. Edited by Gérard Cohen and Jacques Wolfmann. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1989, pages 106–113. ISBN: 978-3-540-46726-7. DOI: 10.1007/BFb0019850.

[Wac+] A. Wachter-Zeh et al. *Lecture Notes for Channel Coding*. Summer 2021. Technical University of Munich.

[WP21] A. Wachter-Zeh and S. Puchinger. “Tutorial ”Code-Based Cryptography””. In: *IEEE Information Theory Workshop 2021* (2021).