UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

GRAU DE MATEMÀTIQUES

Treball final de grau

Acceleration Strategies for Post-Quantum Cryptographic Schemes

Autor: David Farré Gil

Directors: Dr. Xevi Guitart Morales
Dr. Oriol Farràs Ventura

Realitzat a: Departament de
Matemàtiques i Informàtica

Barcelona, 21 de juny de 2020

Abstract

The aim of project is to study the quantum-resistant cryptosystems Classic McEliece and NTRU, revising some of their previous literature and proving some of the main results upon which these cryptosystems are built. We also study the implementation strategies for the acceleration of these schemes.

Finally, we make a comparative study of the reference implementations, considering metrics such as performance and key size.

2010 Mathematics Subject Classification. 11T71, 81P94

Contents

Chapter 1

Introduction

The aim of modern cryptography is the study of communications in the presence of an adversary. For this work, we consider the classical setting for cryptography. In this setting, we have two parties A,B that need to communicate via an insecure channel, where their messages can be intercepted. The notion of message encryption refers to a strategy that ensures that this communication is secure.

Currently there are two main approaches to message encryption: Symmetric Key encryption and Asymmetric Key encryption, also known as Public-key.

In Symmetric key encryption, both parties share a secret key, as well as an encryption algorithm, which is usually well known by third parties. To share a message, the first party encrypts it using the secret key, and the second party can recover the original message by applying the encryption algorithm inversely with the secret key to the encrypted message. This method of encryption has an interesting flaw. The problem with symmetric keys is that both parties must agree on a secret key before they can communicate with each other. If the parties could meet and share the secret key without need to send them as messages, there is no problem. But when the parties cannot communicate by any way besides insecure messages (vulnerable to third parties), they could never begin their communication, since they would need a secure communication to share the keys, but they need the keys to make the communication secure. On the other side, Symmetric key encryption is usually very easy to use and implement, and it also makes it consistently hard for third parties to decrypt a message without knowing the secret key.

In Public-key encryption, the first party generates a key pair: A Secret key and a Public key. They keep to themselves the Secret key, and share the Public key to every party observing the channel. Now, the second party can use the public key to encrypt a message-it is interesting to note that any party can encrypt a message, since the public key is available to everyone. The second party can send

the encrypted message to the first party, who will be able to decrypt the message using their secret key. Other third parties will only have access to the Public key, so they will not be able to decrypt the message. This method provides a clear advantage over Symmetric Key encryption, because it does not require the parties to meet securely to begin the communication. On the other side, these algorithms are usually less efficient and more complicated, making for slow communication, and their safety against attackers is also a lot more subtle.

The security of the public-key cryptographic schemes relies upon mathematical problems, that are assumed to be hard to solve. Currently, the most popular schemes are based on *Rivest-Shamir-Adleman* (RSA) or based on the discrete logarithm problem over elliptic curves. The process of factorizing an arbitrarily large integer is very hard and slow. So as long as any attacker is unable to efficiently factorize integers, encryption via algorithms like RSA cannot be broken without knowing the secret key. Using the existing “classical" computers, all known probabilistic polynomial time algorithms attempting to solve these problems only succeed with very small probability [15]. But what happens if suddenly appeared an algorithm which could easily factorise large integers?

Quantum-Resistant Cryptosystems

In 1994, Peter Shor invented an algorithm - commonly referred as Shor’s algorithm [19] - which reduces the complexity of finding the prime factors of large integers. Even though Shor’s algorithm was a great advance, because it provided an efficient solution to one of the most well known complexity problems in mathematics, it was largely disregarded because it is an algorithm that can only be implemented on a powerful Quantum Computer, something that most considered to be science-fiction.

In recent years, there has been a substantial amount of research on quantum computers - machines that exploit quantum mechanical phenomena to solve mathematical problems that are difficult or intractable for conventional computers. If large-scale quantum computers are ever built, they will be able to break many of the public-key cryptosystems currently in use. This would seriously compromise the confidentiality and integrity of digital communications on the Internet and elsewhere. The goal of post-quantum cryptography (also called quantum-resistant cryptography) is to develop cryptographic systems that are secure against both quantum and classical computers, and can interoperate with existing communications protocols and networks.

The question of when a large-scale quantum computer will be built is a complicated one. While in the past it was less clear that large quantum computers are a physical possibility, many scientists now believe it to be merely a significant

engineering challenge. Some engineers even predict that within the next twenty or so years sufficiently large quantum computers will be built to break essentially all public key schemes currently in use. Historically, it has taken almost two decades to deploy our modern public key cryptography infrastructure. Therefore, regardless of whether we can estimate the exact time of the arrival of the quantum computing era, we must begin now to prepare our information security systems to be able to resist quantum computing [4]. According to this need, several standardization entities around the world are exploring for the new standards. Most of these processes are secret, but at NIST the procedure is transparent and the updates are shared publicly. This allows for communication and exploration from outside the institute to contribute to the process.

NIST contest state

Currently, NIST (National Institute of Standards and Technology) is engaged in a process to evaluate and standardize some quantum-resistant KEMs and digital signatures. In the first place, NIST called for proposals of several algorithms. This first call has already undergone Round 1, in which some of the proposals were discarded and some were unified. On January 30, 2019, Round 2 candidates were announced. These candidates are up to evaluation in order to conclude which submitted KEMs can be considered the best, in order to work on its standardization.

1.1 Key Encapsulation Mechanisms (KEM)

All of the algorithms presented in this document are KEMs. Currently, Public Key Encryption is no longer considered an effective way for communication. Instead we are using KEMs, which use public key encryption algorithms to share a symmetric key between two parties, and this key will be used for communication. One of the parties generates (pk, sk), and shares pk publicly with the other party. This party outputs c and k, sharing c publicly with the first party. The first party now obtains back the key k, considering that it does not fail. After this process, both parties share the key k which can be now used as a symmetric key for Symmetric Key Encryption. Now, the question that remains is how safe is this key that the parties share, namely how unlikely is it for other parties to obtain this secret key k.

A Key Encapsulation Mechanism (KEM) is defined as a triple of algorithms K=(KeyGen, Encaps, Decaps) and a corresponding key space K.[16]

1. The probabilistic key generation algorithm KeyGen() returns a public/secret-key pair (pk, sk).

2. The probabilistic encapsulation algorithm Encaps(pk) takes as input a public key pk and outputs a ciphertext c as well as a key $k \in \mathrm{K}$.

3. The deterministic decapsulation algorithm Decaps(sk, c) takes as input a secret key sk and a ciphertext c an returns a key $k \in \mathrm{K}$ or $\perp$, denoting failure.

A KEM K is ϵ-correct if for all $(sk, pk) \leftarrow$ KeyGen() and $(c, k) \leftarrow$ Encaps(pk), it holds that P[Decaps$(sk, c) \neq k] \leq \epsilon$. We say it is correct if $\epsilon = 0$.

Security of KEMs

The security of modern cryptosystems is based on the assumption that some mathematical problems are hard. For instance, the security of the RSA public-key cryptosystem is based on the security of the so-called RSA problem, which is connected to the factorization problem (see [20]). The security of Classic McEliece and its successors is based on decoding problems, which are considered to be NP. For details on this consideration, see [22]. In the same book, the authors suggest: "no proof of security supporting NTRU is known, and confidence in the security of the scheme is gained primarily from the best currently known attacks".

Since the basis for security on modern cryptosystems is the complexity of the problems, there is no clear approach to studying the security solely based on the hardness of deciphering the encrypted message. Then, the modern approach to security consists on modelling the adversary as an algorithm that runs on polynomial time and has access to some parts of the cryptosystem.

The underlying idea behind the following definitions is that whenever the adversary attacks the cryptosystem (expected to be NP) while running at polynomial time, they cannot gain any advantage towards breaking the cryptosystem, i.e. the best attacks are about as effective as randomly guessing. The different definitions correspond to increasing assumptions about the tools that the adversary has at their disposal.

The security of KEMs is defined in terms of the indistinguishability of the session key against chosen-plaintext (IND-CPA) and chosen-ciphertext (IND-CCA) adversaries.

- In the IND-CPA experiment, the challenger generates $(sk, pk) \leftarrow$ KeyGen(), computes $(c, k_0) \leftarrow$ Encaps(pk), and samples k_1 uniformly at random from the key space. Then, the adversary is given c, k_b (k_b can be k_0 or k_1 with equal probability) and pk, and it is asked to decide whether it received the

key corresponding to c (k_0), or a random value (k_1). The adversary wins if it chooses k_b.

- In the IND-CCA experiment, the setting is the same as before, but now the adversary also has access to the decapsulation oracle, which allows it to obtain $k^* \leftarrow \text{Decaps}(sk, c^*)$, for any $c^* \neq c$ (It knows the result of decrypting every ciphertext except for the one at stake).

The KEM is respectively considered IND-CPA/IND-CCA secure if performing calculations in polynomial time allows the adversary to win with a probability at most $0.5 + \epsilon$, where ϵ becomes negligible when the parameters of the cryptosystem become larger.[16]

In the NIST call, all the proposals are asked to be IND-CCA secure. Also, there are security levels, which relate to the hardness of a key search attack on a block cipher, like AES. That is, the comparison is made by taking into account the best known algorithms that break AES and the algorithms that break the analyzed KEMs. Specifically, NIST considers security levels 1, 3 and 5, which correspond to the security of AES-128, AES-192 and AES-256, respectively.

1.2 This work

In this project we aim to provide a mathematical approach to post-quantum cryptography. We check the correctness of this schemes by only using basic results of abstract algebra. This approach contrast with most of the literature, in which these results are usually proven using extensive additional knowledge in coding theory. Unless stated, the proofs are original and intended for a broader mathematical audience.

Following the evaluation of quantum-resistant KEMs being held by NIST, we aim to contribute to the study and analysis of the cryptosystems by elaborating a summary of the main attributes of schemes as well as some techniques for their acceleration. In this project we will focus on two of the most prominent candidates, Classic McEliece and NTRU.

We believe these candidates show some of the best qualities the NIST might be looking for in this contest. Classic McEliece is based on the original scheme from 1978 by Robert J. McEliece [7], which is considered to be a quantum-resistant cryptosystem. NTRU is a more novel lattice-based cryptosystem, which shows promise, but has less history to back it up. Both schemes take these reliable bases and use them to build a modern KEMs adapted to the needs of the Post-Quantum world, as well as showing some notable advantages regarding reliability and efficiency.

In Chapter 2 we will do a thorough study of the Code Structure of Classic McEliece (section 2.1), as well as the Lattice Structure of NTRU (section 2.2), looking at how the schemes work and proving some of the main results that show that these cryptosystems are indeed correct, and therefore reliable to be used as KEMs.

In Chapter 3 we will be looking at the challenges presented when implementing these cryptosystems, and some of the key solutions to solving them. Specifically, we will be looking at the Berlekamp-Massey algorithm (section 3.2) - the main responsible of Classic McEliece decryption - as well as the Number-Theoretic Transform (section 3.3) - a great tool for finite ring multiplications on NTRU.

In Chapter 4 we will do a comparative study of the reference implementations of both schemes, considering the best candidate according to different metrics: Performance, Key sizes, and Area (Memory) required for Hardware implementations.

Chapter 2

PQ cryptosystems

2.1 Classic McEliece

The McEliece cryptosystem belongs to the family of the code-based cryptosystems. In these cryptosystems, the one-way function associated to this cryptosystem is the addition of errors to codewords of a linear code. The trapdoor of this one-way function is the knowledge of an efficient error correcting algorithm for the considered family of linear codes and the knowledge of a secret matrix permutation. The McEliece cryptosystem is considered the first code-based cryptosystem.[21] Classic McEliece is a reinterpretation of the McEliece cryptosytem, adapting it to the modern necessities of KEMs.

In this section we will discuss the structure of the Goppa code used in the McEliece cryptosystem, proving some of the main results regarding the validity of the cryptosystem. This text is elaborated following the results by E.R. Berlekamp on [1]. The cryptosystem corresponds to the NIST submission [5].

2.1.1 Binary Goppa Code Structure

Let m be any integer. Let $t \geq 2$. Let $g(x) \in F_{2^m}[x]$ be a square-free polynomial of degree t. Let $L = \alpha_1, \ldots, \alpha_n$ a set of elements of F_{2^m} that are not roots of $g(x)$.

The Binary Goppa code of location field F_{2^m}, Goppa polynomial $g(x)$ and length n is defined as the set of vectors in $(F_{2^m})^n$ that satisfy

$$\sum_{\alpha_i \in L} \frac{C_i}{x - \alpha_i} \equiv 0 \; mod \; g(x),$$

where C_i is the coefficient of C in the ith position. These vectors C are codewords of the Binary Goppa Code.

This equation can be viewed as a set of t equations, each corresponding to the powers of x. These equations are linear in respect to C_i.

Theorem 2.1. *The matrix H is a parity check matrix of the Binary Goppa Code*

$$H = \begin{bmatrix} \frac{1}{g(a_1)} & \frac{1}{g(a_2)} & \cdots & \frac{1}{g(a_n)} \\ \frac{a_1}{g(a_1)} & \frac{a_2}{g(a_2)} & \cdots & \frac{a_n}{g(a_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{(a_1)^{t-1}}{g(a_1)} & \frac{(a_2)^{t-1}}{g(a_2)} & \cdots & \frac{(a_n)^{t-1}}{g(a_n)} \end{bmatrix}$$

Proof. Let's begin by writing the code equation as a linear system. To do so, we describe the sum polynomial as a polynomial of degree $t-1$ whose coefficients are linear expressions in C_i.

First we need to obtain $p^i(x) \equiv \frac{1}{x-\alpha_i} \ mod \ g(x)$, where $p^i(x) = \sum_{j=0}^{t-1} p_j^i x^j$

$$\begin{aligned} 1 &\equiv (x-\alpha_i)p^i(x) \equiv (x-\alpha_i)\sum_{j=0}^{t-1} p_j^i x^j \equiv \sum_{j=0}^{t-1} p_j^i x^{j+1} - \sum_{j=0}^{t-1} \alpha_i p_j^i x^j \ mod \ g(x) \\ &\equiv \sum_{j=1}^{t} p_{j-1}^i x^j - \sum_{j=0}^{t-1} \alpha_i p_j^i x^j \equiv -\alpha_i p_0^i + \sum_{j=1}^{t-1} (p_{j-1}^i - \alpha_i p_j^i)x^j + p_{t-1}^i x^t \ mod \ g(x) \end{aligned}$$

Now, we consider $g(x) = \sum_{j=0}^{t} g_j x^j$. Without loss of generality, we can assume the polynomial $g(x)$ is monic. If it were not, we consider $\frac{g(x)}{g_t}$; with this polynomial we will obtain a parity check matrix $g_t * H$. This matrix is equivalent to the matrix H, so the result holds.

Since $g(x) \equiv 0$, $x^t \equiv -\sum_{j=0}^{t-1} g_j x^j$. Also, we call $g^k(x) = \sum_{j=0}^{k} g_j x^j$.

$$\begin{aligned} 1 &\equiv -\alpha_i p_0^i + \sum_{j=1}^{t-1} (p_{j-1}^i - \alpha_i p_j^i)x^j - p_{t-1}^i \sum_{j=0}^{t-1} g_j x^j \ mod \ g(x) \\ &\equiv -\alpha_i p_0^i - p_{t-1}^i g_0 + \sum_{j=1}^{t-1} (p_{j-1}^i - \alpha_i p_j^i - p_{t-1}^i g_j)x^j \ mod \ g(x) \end{aligned}$$

The coefficients of the polynomial $p^i(x)$ are given by the system of equations

$$\begin{bmatrix} -\alpha_i & 0 & \dots & 0 & -g_0 \\ 1 & -\alpha_i & \ddots & \vdots & \vdots \\ 0 & \ddots & \ddots & 0 & \vdots \\ \vdots & \ddots & 1 & -\alpha_i & -g_{t-2} \\ 0 & \dots & 0 & 1 & -\alpha_i - g_{t-1} \end{bmatrix} \begin{bmatrix} p_0^i \\ p_1^i \\ \vdots \\ p_{t-2}^i \\ p_{t-1}^i \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

Now, we solve it by Gaussian elimination. Let Id be the $t \times t$ identity matrix, and

let A be the $t \times (t-1)$ matrix defined as

$$A = \begin{bmatrix} -\alpha_i & 0 & \dots & 0 \\ 0 & -(\alpha_i)^2 & \ddots & \vdots \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & 0 & -(\alpha_i)^{t-1} \\ 0 & \dots & 0 & 0 \end{bmatrix}$$

Then,

$$\left[\begin{array}{ccccc|c} -\alpha_i & 0 & \dots & 0 & -g_0 & 1 \\ 1 & -\alpha_i & \ddots & \vdots & \vdots & 0 \\ 0 & \ddots & \ddots & 0 & \vdots & \vdots \\ \vdots & \ddots & 1 & -\alpha_i & -g_{t-2} & 0 \\ 0 & \dots & 0 & 1 & -\alpha_i - g_{t-1} & 0 \end{array}\right]$$

$$\sim \left[\begin{array}{ccccc|c} -\alpha_i & 0 & \dots & 0 & -g_0 & 1 \\ \alpha_i & -(\alpha_i)^2 & \ddots & \vdots & -g_1\alpha_i & 0 \\ 0 & \ddots & \ddots & 0 & \vdots & \vdots \\ \vdots & \ddots & (\alpha_i)^{t-2} & -(\alpha_i)^{t-1} & -g_{t-2}(\alpha_i)^{t-2} & 0 \\ 0 & \dots & 0 & (\alpha_i)^{t-1} & -(\alpha_i)^t - g_{t-1}(\alpha_i)^{t-1} & 0 \end{array}\right]$$

$$\sim \left[\begin{array}{ccc|c} & & -g_0 & 1 \\ & & -\sum_{j=0}^{1} g_j(\alpha_i)^j & 1 \\ & A & \vdots & \vdots \\ & & -\sum_{j=0}^{t-2} g_j(\alpha_i)^j & 1 \\ & & -(\alpha_i)^t - \sum_{j=0}^{t-1} g_j(\alpha_i)^j & 1 \end{array}\right] \sim \left[\begin{array}{ccc|c} & & -g^0(\alpha_i) & 1 \\ & & -g^1(\alpha_i) & 1 \\ & A & \vdots & \vdots \\ & & -g^{t-2}(\alpha_i) & 1 \\ & & -g^t(\alpha_i) & 1 \end{array}\right]$$

$$\sim \left[\begin{array}{ccc|c} & & -g^0(\alpha_i) & 1 \\ & & -g^1(\alpha_i) & 1 \\ & A & \vdots & \vdots \\ & & -g^{t-2}(\alpha_i) & 1 \\ & & 1 & \frac{-1}{g(\alpha_i)} \end{array}\right] \sim \left[\begin{array}{ccc|c} & & 0 & 1 - \frac{g^0(\alpha_i)}{g(\alpha_i)} \\ & & 0 & 1 - \frac{g^1(\alpha_i)}{g(\alpha_i)} \\ & A & \vdots & \vdots \\ & & 0 & 1 - \frac{g^{t-2}(\alpha_i)}{g(\alpha_i)} \\ & & 1 & \frac{-1}{g(\alpha_i)} \end{array}\right]$$

$$\sim \left[\begin{array}{ccc|c} & & 0 & \frac{1}{g(\alpha_i)}\sum_{k=1}^{t} g_k(\alpha_i)^k \\ & & 0 & \frac{1}{g(\alpha_i)}\sum_{k=2}^{t} g_k(\alpha_i)^k \\ & A & \vdots & \vdots \\ & & 0 & \frac{1}{g(\alpha_i)}\sum_{k=t-1}^{t} g_k(\alpha_i)^k \\ & & 1 & \frac{-1}{g(\alpha_i)} \end{array}\right]$$

$$\sim \left[\begin{array}{ccc|c} & & & \frac{-1}{g(\alpha_i)}\sum_{k=1}^{t} g_k(\alpha_i)^{k-1} \\ & & & \frac{-1}{g(\alpha_i)}\sum_{k=2}^{t} g_k(\alpha_i)^{k-2} \\ & Id & & \vdots \\ & & & \frac{-1}{g(\alpha_i)}\sum_{k=t-1}^{t} g_k(\alpha_i)^{k-t+1} \\ & & & \frac{-1}{g(\alpha_i)} \end{array}\right] \sim \left[\begin{array}{ccc|c} & & & \frac{-1}{g(\alpha_i)}\sum_{k=0}^{t-1} g_{k+1}(\alpha_i)^k \\ & & & \frac{-1}{g(\alpha_i)}\sum_{k=0}^{t-2} g_{k+2}(\alpha_i)^k \\ & Id & & \vdots \\ & & & \frac{-1}{g(\alpha_i)}\sum_{k=0}^{1} g_{k+t-1}(\alpha_i)^k \\ & & & \frac{-1}{g(\alpha_i)} \end{array}\right]$$

In conclusion:

$$p^i(x) = \sum_{j=0}^{t-1} p_j^i x^j = \sum_{j=0}^{t-1} \left(\frac{-1}{g(\alpha_i)} \sum_{k=0}^{t-1-j} g_{k+j+1}(\alpha_i)^k \right) x^j$$

We recall that

$$\sum_{\alpha_i \in L} \frac{C_i}{x - \alpha_i} \equiv \sum_{\alpha_i \in L} p^i(x) C_i \equiv \sum_{\alpha_i \in L} \sum_{j=0}^{t-1} p_j^i x^j C_i \equiv \sum_{j=0}^{t-1} \sum_{\alpha_i \in L} p_j^i C_i x^j \equiv 0 \ mod\ g(x)$$

Then, the linear code is given by equations $\sum_{\alpha_i \in L} p_j^i C_i = 0$ for every j, i.e. H' is a Parity Check Matrix:

$$H' = \begin{bmatrix} p_{t-1}^1 & p_{t-1}^2 & \cdots & p_{t-1}^n \\ \vdots & \vdots & \ddots & \vdots \\ p_1^1 & p_1^2 & \cdots & p_1^n \\ p_0^1 & p_0^2 & \cdots & p_0^n \end{bmatrix}$$

The matrix H can be obtained as an equivalent matrix to H′ via steps of Gaussian reduction, therefore H is indeed a Parity Check Matrix of the Goppa Code. For $j \in \{0, \cdots, t-1\}$, R_j are the rows of the matrix H', and $\hat{R}_j$ are the rows of the matrix H.

$$R_j = \left(p_{t-1-j}^1, \cdots, p_{t-1-j}^n \right) = \left(\frac{-1}{g(\alpha_1)} \sum_{k=0}^{j} g_{k+t-j}(\alpha_1)^k, \ldots, \frac{-1}{g(\alpha_n)} \sum_{k=0}^{j} g_{k+t-j}(\alpha_n)^k \right)$$

$$\hat{R}_j = \left(\frac{(\alpha_1)^j}{g(\alpha_1)}, \ldots, \frac{(\alpha_n)^j}{g(\alpha_n)} \right)$$

For $0 \leq j \leq t-1$, we obtain $\hat{R}_j$ as a linear combination of R_j and $\hat{R}_k$ with $0 \leq k < j$,

$$\hat{R}_j = \left(\sum_{k=0}^{j-1} g_{k+t-j} \hat{R}_k - R_j \right)$$

so the matrix H is indeed equivalent to H'. □

When an error vector $E \in (F_{2^m})^n$ is added to a codeword $C \in (F_{2^m})^n$, the received word R is given by $R = C + E$. Since the code is linear,

$$\sum_{\alpha_i \in L} \frac{R_i}{x - \alpha_i} = \sum_{\alpha_i \in L} \frac{C_i}{x - \alpha_i} + \sum_{\alpha_i \in L} \frac{E_i}{x - \alpha_i}.$$

Since C is a codeword, the first sum on the right side is zero *mod* $g(x)$, therefore

$$\sum_{\alpha_i \in L} \frac{R_i}{x - \alpha_i} \equiv \sum_{\alpha_i \in L} \frac{E_i}{x - \alpha_i} \ mod \ g(x).$$

The syndrome polynomial $S(x)$ is defined as the polynomial of degree $\leq t$ such that

$$S(x) \equiv \sum_{\alpha_i \in L} \frac{R_i}{x - \alpha_i} \equiv \sum_{\alpha_i \in L} \frac{E_i}{x - \alpha_i} \ mod \ g(x).$$

Let $M = \{\alpha_i \in L \mid E_i \neq 0\} \subset L$. Since $E_i = 0$ for the i's that are not in M, the previous sum can be written as:

$$S(x) \equiv \sum_{\alpha_i \in M} \frac{E_i}{x - \alpha_i} \ mod \ g(x).$$

Consider the error-locator polynomial, a polynomial that has as roots the locations of errors in E.

$$\sigma(x) = \prod_{\alpha_i \in M} (x - \alpha_i).$$

Let $\mu(x)$ be a variant of the previous polynomial defined as

$$\mu(x) = \sum_{\alpha_i \in M} E_i \prod_{\alpha_j \in M \setminus \{\alpha_i\}} (x - \alpha_j).$$

Lemma 2.2. *The polynomials $\sigma(x)$ and $\mu(x)$ are relatively prime.*

Proof. It is enough to see that none of the monomials $(x - \alpha_i)$ in $\sigma(x)$ divides the polynomial $\mu(x)$.

Suppose that there exists a $(x - \alpha_k)$ dividing $\mu(x)$. For every $\alpha_k \neq \alpha_i$,

$$(x - \alpha_k) \mid \prod_{\alpha_j \in M \setminus \{\alpha_i\}} (x - \alpha_j).$$

Therefore,

$$(x - \alpha_k) \mid \sum_{\alpha_i \in M \setminus \{\alpha_k\}} E_i \prod_{\alpha_j \in M \setminus \{\alpha_i\}} (x - \alpha_j).$$

Since $(x - \alpha_k)$ divides $\mu(x)$, we also have:

$$(x - \alpha_k) \mid E_k \prod_{\alpha_j \in M \setminus \{\alpha_k\}} (x - \alpha_j).$$

But none of the monomials in the product is $(x - \alpha_k)$, and E_k is nonzero by definition of M, so this is a contradiction. □

Next, we want to see that the errors in the vector E are given by the polynomials $\sigma(x)$ and $\mu(x)$.

Proposition 2.3. *The coordinates E_i of the vector are*

$$0 \text{ if } \sigma(\alpha_i) \neq 0 \quad \text{and} \quad \frac{\mu(\alpha_i)}{\sigma'(\alpha_i)} \text{ if } \sigma(\alpha_i) = 0,$$

where $\sigma'(x)$ is the formal derivative of $\sigma(x)$.

Proof. By the definition of $\sigma(x)$, for any $E_i \neq 0$, $\sigma(\alpha_i) = 0$, so the first affirmation is clear.
We evaluate the formal derivative of $\sigma(x)$:

$$\sigma'(x) = \sum_{\alpha_i \in M} \prod_{\alpha_j \in M \setminus \{\alpha_i\}} (x - \alpha_j).$$

Notice that for each k, the product vanishes for every i except when $i = k$.

$$\mu(\alpha_k) = \sum_{\alpha_i \in M} E_i \prod_{\alpha_j \in M \setminus \{\alpha_i\}} (\alpha_k - \alpha_j) = E_k \prod_{\alpha_j \in M \setminus \{\alpha_k\}} (\alpha_k - \alpha_j).$$

$$\sigma'(\alpha_k) = \sum_{\alpha_i \in M} \prod_{\alpha_j \in M \setminus \{\alpha_i\}} (\alpha_k - \alpha_j) = \prod_{\alpha_j \in M \setminus \{\alpha_k\}} (\alpha_k - \alpha_j).$$

Combining the two equations, we obtain $\mu(\alpha_k) = E_k \sigma'(\alpha_k)$, which proves the second affirmation. □

So far, we've seen that given the polynomials $\sigma(x), \mu(x)$ we can recover the error vector E and therefore the codeword C. Now, we need to prove that given the syndrome polynomial $S(x)$ we can obtain $\sigma(x), \mu(x)$.

By multiplying the defining expressions for $S(x)$ and $\sigma(x)$, we obtain $\mu(x)$

$$S(x)\sigma(x) \equiv \sum_{\alpha_i \in M} \frac{E_i}{x - \alpha_i} \prod_{\alpha_j \in M} (x - \alpha_j) \equiv \sum_{\alpha_i \in M} E_i \prod_{\alpha_j \in M \setminus \{\alpha_i\}} (x - \alpha_j) \equiv \mu(x) \ mod \ g(x).$$

If we consider the representative $p(x)$ of

$$S(x)\sigma(x) - \mu(x) \ mod \ g(x)$$

such that it is of degree lower or equal than t, we obtain a linear system of t equations with the coefficients of $\sigma(x), \mu(x)$ as unknowns, given by the equality between polynomials $p(x) = 0$.

Theorem 2.4. *There is a t-error correcting code for the Goppa code with polynomial $g(x)$.*

Proof. (Based on the proof on [1]) To prove that we can decode up to t errors, it is enough to show that the linear system has no more than one solution when the degrees of $\sigma(x), \mu(x)$ are sufficiently small. Let's suppose given two different solutions:

$$S(x)\sigma(x) \equiv \mu(x) \ mod \ g(x)$$

$$S(x)\sigma^1(x) \equiv \mu^1(x) \ mod \ g(x)$$

We observe that $\sigma(x)$ is invertible $mod \ g(x)$. If it were not, $\sigma(x)$ would share a common factor with $g(x)$. Then this factor should also divide $\mu(x)$, contradicting the fact that $\sigma(x), \mu(x)$ are relatively prime. Then we can obtain the two following relations:

$$S(x) \equiv \frac{\mu(x)}{\sigma(x)} \ mod \ g(x)$$

$$S(x) \equiv \frac{\mu^1(x)}{\sigma^1(x)} \ mod \ g(x)$$

From these, we obtain

$$\sigma(x)\mu^1(x) \equiv \sigma^1(x)\mu(x) \ mod \ g(x)$$

First, we observe that $\sigma'(x) = \mu(x)$ Then, we observe that $x\sigma'(x)$ is the odd part of $\sigma(x)$. We call $\hat{\sigma(x)}$ the even part of $\sigma(x)$. With this notation:

$$(\hat{\sigma}(x) + x\sigma'(x))\sigma^{1'}(x) \equiv (\hat{\sigma^1}(x) + x\sigma^{1'}(x))\sigma'(x) \ mod \ g(x)$$

$$\hat{\sigma}(x)\sigma^{1'}(x) + \hat{\sigma}^1(x)\sigma'(x) \equiv 0 \ mod \ g(x)$$

Since $\sigma'(x)$ is even, all the polynomials in the last expression are even. Then,

$$\hat{\sigma}(x)\sigma^{1'}(x) + \hat{\sigma}^1(x)\sigma'(x) \equiv 0 \ mod \ g^2(x)$$

Since $g^2(x)$ has degree $2t$, if $\sigma(x), \sigma^1(x)$ have degrees lower or equal than t, we obtain

$$\hat{\sigma}(x)\sigma^{1'}(x) = \hat{\sigma}^1(x)\sigma'(x)$$

Since $\sigma(x), \sigma'(x)$ are relatively prime, we obtain $\sigma(x) = \sigma^1(x)$. In conclusion the code can recover an error vector of weight up to t. □

2.1.2 Cryptosystem algorithm

The system parameters are positive integers m, n, t such that $n \leq 2^m$, and $t \geq 2$. We define $q = 2^m$ and, $k = n - mt$.

1. The secret key has two parts: First, a sequence $a_1, \ldots, a_n$ of distinct elements of F_q. Second, a square-free polynomial $g(x) \in F_q[x]$ of degree t, such that $\forall i \in \{1, \ldots, n\}, g(a_i) \neq 0$.

2. For the public key we compute the following $t \times n$ matrix with coefficients in F_q:

$$H = \begin{bmatrix} \frac{1}{g(a_1)} & \frac{1}{g(a_2)} & \cdots & \frac{1}{g(a_n)} \\ \frac{a_1}{g(a_1)} & \frac{a_2}{g(a_2)} & \cdots & \frac{a_n}{g(a_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{(a_1)^{t-1}}{g(a_1)} & \frac{(a_2)^{t-1}}{g(a_2)} & \cdots & \frac{(a_n)^{t-1}}{g(a_n)} \end{bmatrix}$$

3. Then we extend the previous matrix by writing the elements in F_q as column vectors of F_2^m, $\frac{a_j^{i-1}}{g(a_j)} \rightarrow (c_0, \ldots, c_{m-1})$, using the polynomial expression to represent an element of F_q.

4. Lastly, we perform complete Gaussian elimination on a $n \times mt$ binary matrix, obtaining (I, T), where I is the identity matrix of size mt and T is a $mt \times k$ matrix. The public key is the matrix T.

For the encryption, we take a vector $e \in F_2^n$ of weight t (t coefficients of e are 1). The ciphertext is obtained as $C_0 = (I, T)e \in F_2^n$, of size mt.

1. For the decryption, given C_0 of dim $n - k$, we consider $V = (C_0, 0, \ldots, 0)$ of dim n. We find the unique c in the Goppa Code at distance $\leq t$ from v. Finally, we consider $e = v + c$, and check if weight$(e) = t$ and $C_0 = (I, T)e$.

2. To find the c, we need to compute the double-size $2t \times n$ parity check matrix

$$H^{(2)} = \begin{bmatrix} \frac{1}{g^2(a_1)} & \frac{1}{g^2(a_2)} & \cdots & \frac{1}{g^2(a_n)} \\ \frac{a_1}{g^2(a_1)} & \frac{a_2}{g^2(a_2)} & \cdots & \frac{a_n}{g^2(a_n)} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{(a_1)^{2t-1}}{g^2(a_1)} & \frac{(a_2)^{2t-1}}{g^2(a_2)} & \cdots & \frac{(a_n)^{2t-1}}{g^2(a_n)} \end{bmatrix}$$

3. Similarly to during the key generation, we extend the matrix $H^{(2)}$ to a $2mt \times n$ matrix $H'^{(2)}$ by writing the elements in F_q as column vectors of F_2^m. We compute the double-size syndrome $S^{(2)} = H'^{(2)} \times C_0$

4. Using the Berlekamp-Massey algorithm we obtain the error-locator polynomial $\sigma(x)$ from $S^{(2)}$.

5. Evaluating $\sigma(x)$ at $a_1, \ldots, a_n$, we obtain t roots, which represent the non-zero coefficients of the error vector e. Given that non-zero coefficients are 1, the vector can be recovered accordingly.

2.2 NTRU

NTRU was originally proposed by Hoffstein, Pipher and Silverman in 1996. The scheme was originally presented using polynomial rings. The security of the NTRU scheme is based on the hardness of solving a system of linear equations. The hardness of the underlying mathematical problem has been studied for many years, and can also be formulated in terms of lattices. For this reason, NTRU is usually considered a lattice-based cryptographic scheme.[21]

In the following section we will give some preliminary notions about the structure of the lattice used in the NTRU crptosystem, and then explain the algorithm used in the NTRU submission.

2.2.1 Lattice Structure and Preliminaries

Let n be an odd prime such that 2 and 3 are of order $n-1$ in $(\mathbb{Z}/n)^*$. Let q be a power of 2. We consider the polynomials

$$\Phi_1 = x - 1, \ \Phi_n = \frac{x^n - 1}{x - 1} = \sum_{i=0}^{n-1} x^i$$

We consider several quotient rings:

$$R = \mathbb{Z}[x]/(\Phi_1 \Phi_n), \quad S = \mathbb{Z}[x]/(\Phi_n)$$

$$R/3 = \mathbb{Z}[x]/(3, \Phi_1\Phi_n), \quad R/q = \mathbb{Z}[x]/(q, \Phi_1\Phi_n),$$
$$S/3 = \mathbb{Z}[x]/(3, \Phi_n), \quad S/q = \mathbb{Z}[x]/(q, \Phi_n).$$

For the coefficients over finite rings, we consider the following canonical representatives:

$$\mathbb{Z}/3 = \{-1, 0, 1\}, \quad \mathbb{Z}/q = \{-q/2, \ldots, q/2 - 1\}.$$

Then, a polynomial in the previous rings will be a canonical representative if all of its coefficients are canonical representatives of the corresponding ring.

We consider f, g two polynomials that are canonical representatives of elements in $S/3$, i.e. two polynomials of degree at most $n-2$ with coefficients in $\{-1, 0, 1\}$. Furthermore, the polynomial g must have exactly $(q/16 - 1)$ coefficients equal to 1, and $(q/16 - 1)$ coefficients equal to -1.

The NTRU lattice is defined by the polynomials f, g. Now, we build the decoder of the lattice.

- We consider f_q the canonical representative of f^{-1} in S/q.
- We consider h the canonical representative of $3 * g * f$ in R/q.
- We consider h_q the canonical representative of h^{-1} in S/q.
- We consider f_3 the canonical representative of f^{-1} in $S/3$.

Then the decoder for the lattice is (f, f_3, h_q).
Given a message m and an error r as polynomials in $S/3$, and m having exactly $(q/16 - 1)$ coefficients equal to 1, and $(q/16 - 1)$ coefficients equal to -1, we consider the ciphertext c as the canonical representative of $r * h + m$ in R/q.
To decrypt the message, we consider a the canonical representative of $c * f$ in R/q.

- We consider m' the canonical representative of $a * f_3$ in $S/3$.
- We consider r' the canonical representative of $(c - m') * h_q$ in S/q.

Lemma 2.5. *Given two polynomials $a(x), b(x)$ of degree $n-1$, their product is:*

$$a(x)b(x) = \sum_{k=0}^{n-1} a_k x^k \sum_{j=0}^{n-1} b_j x^j = \sum_{k=0}^{n-1} \left(\sum_{j=0}^{k} a_j b_{k-j} + x^n \sum_{j=k+1}^{n-1} a_j b_{k+n-j} \right) x^k$$

Proof.

$$a(x)b(x) = \sum_{k=0}^{n-1} a_k x^k \sum_{j=0}^{n-1} b_j x^j = \sum_{k=0}^{n-1} \sum_{j=0}^{k} a_j b_{k-j} x^k + \sum_{k=n}^{2n-1} \sum_{j=k-n+1}^{n-1} a_j b_{k-j} x^k$$

To obtain the last expression, we group the coefficients for each x^k. These are obtained by multiplying coefficients of both polynomials such that their indexes add up to k, hence the products $a_j b_{k-j}$. Now it only remains to check for which j this product makes sense, i.e. we need to ensure $0 \leq j \leq n-1$, $0 \leq k-j \leq n-1$. The second condition is equivalent to $k-(n-1) \leq j \leq k$.

- When $k \leq n-1$, $k-(n-1) \leq 0 \leq j \leq k \leq n-1$, so j runs from 0 to k.
- When $n \leq k$, $0 \leq k-(n-1) \leq j \leq n-1 \leq k$, so j runs from $k-n+1$ to $n-1$.
- Also, for $k \geq 2n-1$, all coefficients are 0, but for a better expression we will keep writing term $2n-1$.

Now, we change the indices in the second addend as $k' = k-n$

$$\sum_{k=n}^{2n-1} \sum_{j=k-n+1}^{n-1} a_j b_{k-j} x^k = \sum_{k'=0}^{n-1} \sum_{j=k'+1}^{n-1} a_j b_{k'+n-j} x^{k'+n} = x^n \sum_{k'=0}^{n-1} \sum_{j=k'+1}^{n-1} a_j b_{k'+n-j} x^{k'}$$

Gathering these results we obtain:

$$\begin{aligned} a(x)b(x) &= \sum_{k=0}^{n-1} \sum_{j=0}^{k} a_j b_{k-j} x^k + x^n \sum_{k=0}^{n-1} \sum_{j=k+1}^{n-1} a_j b_{k+n-j} x^k \\ &= \sum_{k=0}^{n-1} \left(\sum_{j=0}^{k} a_j b_{k-j} + x^n \sum_{j=k+1}^{n-1} a_j b_{k+n-j} \right) x^k \end{aligned}$$

□

Proposition 2.6. *The output message is indeed the same as the original, i.e.* $(r, m) = (r', m')$.

Proof. Given the definitions for a and c, we study m' in terms of previously defined polynomials, such as r, m and the polynomials in the secret key.

$$c \equiv r * h + m \text{ in } R/q$$

$$a \equiv c * f \equiv (r * h + m) * f \equiv r * 3 * g * f_q * f + m * f \equiv r * 3 * g + m * f \text{ in } R/q$$

Hence, a is equal to

$$a = r * 3 * g + m * f + (x^n - 1) * d(x) + q * b(x)$$

for some polynomials $d(x)$ and $b(x)$. Due to the construction of the polynomials in the definition of a, we can find $d(x)$ such that $b(x) = 0$.

Notice that m, f, are two polynomials with coefficients over degree $n-2$ with coefficients over $\{-1,0,1\}$. Furthermore, m has exactly $(q/16-1)$ coefficients equal to 1, and $(q/16-1)$ coefficients equal to -1; overall $(q/8-2)$ non-zero coefficients. Let's consider their multiplication. For practical reasons we consider their coefficients up to $n-1$, even though they are 0. By Lemma 2.5,

$$m * f = \sum_{k=0}^{n-1} m_k x^k \sum_{j=0}^{n-1} f_j x^j = \sum_{k=0}^{n-1} \left(\sum_{j=0}^{k} m_j f_{k-j} + x^n \sum_{j=k+1}^{n-1} m_j f_{k+n-j} \right) x^k$$

Considering modular reduction in R/q using $x^n - 1$ (which affects the polynomial $d(x)$), we have $x^n = 1$.

$$m * f - (x^n - 1) * d_1(x) = \sum_{k=0}^{n-1} \left(\sum_{j=0}^{k} m_j f_{k-j} + \sum_{j=k+1}^{n-1} m_j f_{k+n-j} \right) x^k$$

Then, this polynomial is of degree $n-1$, and its coefficients are obtained as a sum of n elements over $\{-1,0,1\}$. Since at most $(q/8-2)$ of these elements are non-zero, the coefficients must be in the range $\{-q/8+2,\ldots,q/8-2\}$. Similarly, g, r are defined like m, f, respectively. Then, we can also find

$$g * r - (x^n - 1) * d_2(x)$$

of degree $n-1$ with coefficients in the same range.

$$3 * g * r - 3 * (x^n - 1) * d_2(x)$$

will have coefficients over $\{-3q/8+6,\ldots,3q/8-6\}$, and their sum

$$r * 3 * g + m * f - (x^n - 1) * (d_1(x) + 3d_2)(x)$$

will have coefficients over $\{-q/2+8,\ldots,q/2-8\}$. Also, since it is of degree $n-1$ it is a canonical representative, so it is indeed a. Then $b(x) = 0$.

Now we consider m' using the definition we found for a:

$$\begin{aligned} m' &\equiv a * f_3 \equiv (r * 3 * g + m * f + (x^n - 1) * d(x)) * f_3 \\ &\equiv r * 0 * g * f_3 + m * f * f_3 + 0 * d(x) * f_3 \equiv m \text{ in } S/3 \end{aligned}$$

Then, we obtain that $m' \equiv m$ in $S/3$, and since both are canonical representatives, they are indeed the same.

Since we have c defined in R/q ($mod(q, \phi_1\phi_n)$), and both $q, \phi_1\phi_n$ are actually 0 in S/q, c has the same definition in S/q.

$$r' \equiv (c - m') * h_q \equiv ((r * h + m) - m) * h_q \equiv r * h * h_q \equiv r \text{ in } S/q$$

Even though r is defined in $S/3$, since its coefficients are over $\{-1,0,1\}$, it is also a canonical representative in S/q. Then, $r' \equiv r$ in S/q and both are canonical representatives, so they are indeed the same. □

2.2.2 Cryptosystem algorithm

The implementation of the algorithm is the following, as seen on [6].

1. Generating f and g two polynomials of degree n with 1, 0 and -1.
2. Inverting f $\text{mod}(3, \Phi_n)$; we obtain f_p.
3. Multiplying $3g * f \ \text{mod}(q, \Phi_n)$; we obtain v_0.
4. Inverting $v_0 \ \text{mod}(q, \Phi_n)$; we obtain v_1.
5. Multiplying $v_1 * 3g * 3g \ \text{mod}(q, \Phi_1 * \Phi_n)$; we obtain h.
6. Multiplying $v_1 * f * f \ \text{mod}(q, \Phi_1 * \Phi_n)$; we obtain h_q.
7. The secret key is f, f_p, h_q. The public key is h.

In order to obtain the inverses, we use the Almost Inverse algorithm[17]. For $\text{mod}(q)$, the Almost Inverse algorithm is performed $\text{mod}(2)$, and later we have to calculate some polynomial multiplications.
First, we perform a subtle transformation $Lift$ to the message m_0 to obtain m_1 (sometimes this transformation is just the identity, and otherwise they're simple operations $\text{mod}(3, \Phi_n)$). Then, the ciphertext is obtained as follows:

$$c = r * h + m_1 \ \text{mod}(q, \Phi_1 * \Phi_n)$$

Where r is a random noise.

1. Multiplying $c * f \ \text{mod}(q, \Phi_1 * \Phi_n)$; we obtain v_1.
2. Multiplying $v_1 * f_p \ \text{mod}(3, \Phi_n)$; we obtain m_0.
3. Performing the subtle transformation $Lift$ to the message m_0 to obtain m_1
4. We obtain the noise $r = (c - m_1) * h_q \ \text{mod}(q, \Phi_n)$.
5. Lastly, we have to check that both the message and the noise are valid, otherwise, we consider the process has failed.

Chapter 3

Acceleration of PQ schemes

As we have discussed on previous chapters, the purpose of these cryptosystems is to make it hard for an adversary who does not know the secret key to decrypt the message. While this is needed in order to ensure the security of the cryptosystems, it is also interesting for the Key Generation, Encryption and Decryption algorithms to be as efficient as possible.
This acceleration aims to reduce the resources required for the parties A and B to share messages. If the algorithm can be implemented more efficiently, the communication will be faster and will require less powerful computers.

3.1 General acceleration techniques

In all cryptosystems, there are some acceleration strategies that are commonly used. These strategies are related to basic operations commonly performed in cryptosystems. Binary Matrix-Vector multiplication is mainly used during the encryption and decryption of the Classic McEliece cryptosystem when obtaining the syndrome. In both cryptosystems the elements are in finite rings, so we need an efficient way to invert these elements.

Binary Matrix-Vector multiplication

First, it is interesting to notice that binary vectors, and by extension binary matrices, are very easy for computers to work with. A binary vector of size 8 can be stored precisely as a byte, with each bit corresponding to each component of the vector. Bigger vectors can be stored in the same way using consecutive bytes. This allows for vector addition to be performed using the operation XOR $\oplus$.

Notice that indeed $0 \oplus 0 = 0$, $0 \oplus 1 = 1 \oplus 0 = 1$, $1 \oplus 1 = 0$.

Now, regarding the Matrix-Vector multiplication, we observe the usual procedure, which requires nm multiplications and $(m-1)n$ additions of binary elements.

$$\begin{bmatrix} a_1^1 & \dots & a_1^m \\ \vdots & \ddots & \vdots \\ a_n^1 & \dots & a_n^m \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{m} a_1^j b_j \\ \vdots \\ \sum_{j=1}^{m} a_n^j b_j \end{bmatrix}$$

The following approach considers the matrix multiplication as a linear combination of the columns of the matrix. The advantage this procedure provides is that the coefficients b_j tell us whether we have to add the column, depending on whether it is 1 or 0. In this case, we need to check m coefficients and add up to m vectors of size n. Since these vectors are stored as we mentioned before, we only need to perform at most $m\frac{n}{8}$ XOR.

$$\begin{bmatrix} a_1^1 & \dots & a_1^m \\ \vdots & \ddots & \vdots \\ a_n^1 & \dots & a_n^m \end{bmatrix} \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} = b_1 \begin{bmatrix} a_1^1 \\ \vdots \\ a_n^1 \end{bmatrix} + \dots + b_m \begin{bmatrix} a_1^m \\ \vdots \\ a_n^m \end{bmatrix}$$

This has several advantages. The main one, is that the actual additions are performed much more efficiently, since in the previous method there is no clear way to benefit from the XOR addition. Also, in the second method we only "multiply" once by b_j, rather than multiplying at each row. Also, every 0 in the vector means one less addition we need to perform, while 0s do not provide any benefit in the first method.

Inverting elements on a Finite Field

To invert elements over a Finite Field F_{p^m}, we use that $(F_{p^m})^*$ is a cyclic group. Also, when G is a cyclic group of order n, $\forall a \in G, a^n = 1$. Then, $a^{n-1} = a^{-1}$. For F_{p^m}, this means that $\forall a \in (F_{p^m})^*, a^{p^m-2} = a^{-1}$, because $(F_{p^m})^*$ is of order $p^m - 1$. This method gives a way to invert elements over F_{p^m} using only multiplication. This method can be further improved by using efficient exponentiation. Different methods can be used depending on the specific exponentiation we want to perform. Exponentiation by squaring is a general method defined by the following recursive algorithm:

$$Power(a,b) = \begin{cases} 1, & \text{if } b = 0 \\ a * Power(a, b-1), & \text{if } b \text{ is odd} \\ Power(a, b/2)^2, & \text{if } b \text{ is even} \end{cases}$$

3.2 Berlekamp-Massey Algorithm

The Berlekamp-Massey Algorithm is the key to decryption on a Binary Goppa Code. The algorithm takes the syndrome polynomial $S(x)$ of the error vector, and returns the error locator polynomial $\sigma(x)$, which can later be used to find the error vector. This formulation of the algorithm is based on the one presented by James L. Massey on [8].

To begin the algorithm, we require some auxiliary variables. We initialize $\sigma(x) = 1, \tau(x) = 1, m = 1, d = 0, b = 1, k = 0, \Delta = 0$

1. When $k = n$, the algorithm has finished. Otherwise, $\Delta = S_n + \sum_{i=1}^{d} \sigma_i S_{k-i}$
2. If $\Delta = 0$, we add one to m, and end the step, going to 5.
3. If $\Delta \neq 0$ and $d > \frac{k}{2}$, we change $\sigma(x)$ to $\sigma(x) - \frac{\Delta}{b} x^m \tau(x)$, then we add one to m, and end the step, going to 5.
4. If $\Delta \neq 0$ and $d \leq \frac{k}{2}$, we store $\sigma(x)$ in an auxiliary vector, then change $\sigma(x)$ to $\sigma(x) - \frac{\Delta}{b} x^m \tau(x)$, and then change $\tau(x)$ to the auxiliary vector (The previous $\sigma(x)$). We reset $m = 1$, we set $b = \Delta$, and $d = k + 1 - d$, and end the step, going to 5.
5. We increase the step k by one and return to step 1.

3.3 Number-Theoretic Transform

In the NTRU algorithm, we need to multiply polynomials over $\mathbb{Z}_m$, which is a very costly operation. Then, we are interested in an algorithm that allows us to more efficiently calculate these multiplications. This can be found using the Number-Theoretic Transform following the results presented in [14].

The Number-Theoretic Transform (NTT), also known as the Discrete Fourier Transform, is a generalization of the Fourier Transform for finite fields.
We consider the finite ring $\mathbb{Z}_{q^n}$, where q is a prime number and n is a power of two such that $q \equiv 1 \ mod \ n$. $\mathbb{Z}_{q^n} \cong \mathbb{Z}_q [x] / < f(x) >$, where $f(x)$ is any irreducible polynomial of degree n over $\mathbb{Z}_q$. Then, the elements of $\mathbb{Z}_{q^n}$ are represented as polynomials over $\mathbb{Z}_q$ of degree at most $n - 1$, with the relation $f(x) = 0$.

Given a polynomial $a(x) = \sum_{k=0}^{n-1} a_k x^k$ with coefficients over $\mathbb{Z}_q$, we define its Number-Theoretic Transform as the polynomial $NTT_\omega(a) = \sum_{i=0}^{n-1} A_i x^i$, where $A_i = \sum_{k=0}^{n-1} a_k (\omega^i)^k$, and ω is a primitive nth root of the unit in $\mathbb{Z}_q$.

Lemma 3.1. *There exist primitive nth roots of unity in $\mathbb{Z}_q$, rather than in an arbitrary extension field.*

Proof. Given $q \equiv 1 \; mod \; n$, we know that $q - 1 = n * k$ for some (positive) integer k.

Since $(\mathbb{Z}_q)^*$ is a cyclic group, there exists an element a of order nk in $(\mathbb{Z}_q)^*$, i.e. a is an nkth primitive root of unity. Then, a^k is an nth primitive root of the unity in $\mathbb{Z}_q$. □

Lemma 3.2. *Given the transformation NTT_ω, there is an inverse transformation $INTT_\omega$ such that $INTT_\omega \circ NTT_\omega = Id$. Furthermore, $INTT_\omega = \frac{1}{n} NTT_{\omega^{-1}}$*

Proof.

$$\text{Given } NTT_\omega(a) = \sum_{i=0}^{n-1} \sum_{k=0}^{n-1} a_k (\omega^i)^k x^i, \text{ consider the composition}$$

$$NTT_{\omega^{-1}}(NTT_\omega(a)) = \sum_{i=0}^{n-1} \sum_{k=0}^{n-1} \left(\sum_{j=0}^{n-1} a_j (\omega^k)^j \right) (\omega^{-i})^k x^i$$

Now let's study the coefficients b_i of the resulting polynomial:

$$b_i = \sum_{k=0}^{n-1} \left(\sum_{j=0}^{n-1} a_j (\omega^k)^j \right) (\omega^{-i})^k = \sum_{k=0}^{n-1} \left(\sum_{j=0}^{n-1} a_j (\omega^k)^{i-j} \right) = \sum_{j=0}^{n-1} a_j \left(\sum_{k=0}^{n-1} (\omega^{i-j})^k \right)$$

$$= a_i \left(\sum_{k=0}^{n-1} (\omega^0)^k \right) + \sum_{j \neq i} a_j \left(\sum_{k=0}^{n-1} (\omega^{i-j})^k \right) = a_i \left(\sum_{k=0}^{n-1} 1 \right) + \sum_{j \neq i} a_j \left(\frac{(\omega^{i-j})^n - 1}{(\omega^{i-j}) - 1} \right)$$

$$= na_i + \sum_{j \neq i} a_j \left(\frac{(\omega^n)^{i-j} - 1}{(\omega^{i-j}) - 1} \right) = na_i + \sum_{j \neq i} a_j \left(\frac{1 - 1}{(\omega^{i-j}) - 1} \right) = na_i$$

Notice that when $i \neq j$, $\omega^{i-j} \neq 1$ because $-n < i - j < n$, and ω is of order n. Then, the $INTT_\omega$ is indeed the inverse of NTT_ω.

$$\frac{1}{n} NTT_{\omega^{-1}}(NTT_\omega(a)) = \frac{1}{n} \sum_{i=0}^{n-1} b_i x^i = \frac{1}{n} \sum_{i=0}^{n-1} na_i x^i = \sum_{i=0}^{n-1} a_i x^i = a(x)$$

□

As we can see in the previous Lemma, the key requirement for the NTT to be invertible is that ω is of order n. Then, the definition of the NTT can be extended to more general situations.

Specifically, the definition holds for finite rings $\mathbb{Z}_q[x] / < f(x) >$ where the polynomial $f(x)$ is not irreducible. Similarly, the ring over the polynomial is defined doesn't have to be a field, i.e. the definition still holds for finite rings $\mathbb{Z}_m[x] / < f(x) >$ where m is not necessarily a prime number and $f(x)$ is not necessarily irreducible, as long as n is invertible in $\mathbb{Z}_m$ and there exists an nth primitive root of unity. These two extensions to the definition can be made because there are no other inversions in the process (besides the primitive root, which is guaranteed to be invertible), so having Zero Divisors does not affect the procedure.

Optimizations for the NTT implementation

We assume given ω, ω^{-1}, and a polynomial $a(x)$ expressed in vector notation $(a_0, \ldots, a_{n-1})$. All the elements a_i, ω, ω^{-1} are in $\mathbb{Z}_q$.

Then, the algorithm computes $A = NTT_\omega(a)$.[14]

1. We consider $\hat{a}$ the vector such that $\hat{a}_k = a_{n-1-k}$. The vector A is also a vector of size n with elements over $\mathbb{Z}_q$. At the beginning it is initialized to 0
2. For every i from 0 to $\log_2 n - 1$, we compute the following:
3. For every j from 0 to $n/2 - 1$

$$P_{ij} = \left\lfloor \frac{j}{2^{\log_2 n-1-i}} \right\rfloor * 2^{\log_2 n-1-i}$$

$$A_j = \hat{a}_{2j} + \hat{a}_{2j+1}\omega^{P_{ij}} \mod q$$

$$A_{j+\frac{n}{2}} = \hat{a}_{2j} - \hat{a}_{2j+1}\omega^{P_{ij}} \mod q$$

Polynomial multiplication using NTT

Depending on the polynomial $f(x)$ defining the ring, this Transform can be used to simplify multiplications. Specifically, we present a method that performs these multiplications when $f(x) = x^n \pm 1$. In the following section we study the case when $f(x) = x^n + 1$. The case $f(x) = x^n - 1$ is similar, but the steps where we apply the variable change are skipped.

For this case we need to introduce the square root of ω, so it is also necessary that there is a $2n$th primitive root of unity (notice that in the case when q is prime, this means $q \equiv 1 \ mod \ 2n$). This root is required to "correct" the sign that comes from the modular reduction (in the case $x^n - 1$, summands stay positive because $x^n = 1$).

We consider given $a(x) = \sum_{k=0}^{n-1} a_k x^k, b(x) = \sum_{k=0}^{n-1} b_k x^k$ polynomials, and we want to obtain the multiplication $a(x)b(x)$ in $\mathbb{Z}_q / < x^n + 1 >$. We also consider given ω, ω^{-1}, and ϕ a square root of ω. We will also need to precompute n^{-1}, ϕ^{-1}; the inverses of these elements *mod* q.[14]

1. We consider $\bar{a}(x) = a(\phi x)$. In practice, this means $\bar{a}_i = a_i \phi^i \ mod \ q$. Similarly for $\bar{b}(x) = b(\phi x)$
2. Then, we consider $\bar{A} = NTT_\omega(\bar{a})$, $\bar{B} = NTT_\omega(\bar{b})$
3. Now that we have the Transforms of a, b, we multiply their transforms in order to obtain the transform of the product. $\bar{C} = \bar{A} \cdot \bar{B}$, i.e. $\bar{C}_i = \bar{A}_i \bar{B}_i \ mod \ q$.

4. In order to recover the product, we need to apply the inverse NTT. $\hat{c} = iNTT_{\omega}(\bar{C}) = n^{-1} * NTT_{\omega^{-1}}(\bar{C})$
5. Finally, we obtain $c(x) = \bar{c}(\phi^{-1}x)$ the product of $a(x), b(x)$. In practice, this means $c_i = \bar{c}_i\phi^{-i}$ *mod* q

Lemma 3.3. *Given two polynomials $a(x), b(x)$ in $\mathbb{Z}_q/\langle x^n+1\rangle$ and the change of variables $\Phi : x \to \phi x$, the transformation of the product is the (scalar) product of the transformations.*

$$NTT_{\omega}(\Phi\left[a(x)b(x)\right]) = NTT_{\omega}(a(\phi x)) \cdot NTT_{\omega}(b(\phi x))$$

Proof. Since x^n+1 is the polynomial describing the ring, we have $x^n = -1$. Together with Lemma 2.5, we obtain:

$$\begin{aligned} a(x)b(x) &= \sum_{k=0}^{n-1} a_k x^k \sum_{j=0}^{n-1} b_j x^j = \sum_{k=0}^{n-1}\left(\sum_{j=0}^{k} a_j b_{k-j} + x^n \sum_{j=k+1}^{n-1} a_j b_{k+n-j}\right) x^k \\ &= \sum_{k=0}^{n-1}\left(\sum_{j=0}^{k} a_j b_{k-j} - \sum_{j=k+1}^{n-1} a_j b_{k+n-j}\right) x^k \end{aligned}$$

Now we obtain the product with the changed variable. Since ϕ is a square root of ω, we know $\phi^{2n} = \omega^n = 1$. If $\phi^n = 1, 1 = (\phi^2)^{\frac{n}{2}} = \omega^{\frac{n}{2}}$. But this is a contradiction, because ω is of order n. Then, ϕ is of order $2n$, and ϕ^n is a square root of unity different than 1, so it is -1.

$$\begin{aligned} \Phi\left[a(x)b(x)\right] &= \sum_{k=0}^{n-1}\left(\sum_{j=0}^{k} a_j b_{k-j} - \sum_{j=k+1}^{n-1} a_j b_{k+n-j}\right)\phi^k x^k \\ &= \sum_{k=0}^{n-1}\left(\sum_{j=0}^{k} a_j b_{k-j}\phi^k - \sum_{j=k+1}^{n-1} a_j b_{k+n-j}\phi^k\right) x^k \\ &= \sum_{k=0}^{n-1}\left(\sum_{j=0}^{k} a_j\phi^j b_{k-j}\phi^{k-j} - \sum_{j=k+1}^{n-1} \phi^{-n} a_j\phi^j b_{k+n-j}\phi^{k+n-j}\right) x^k \\ &= \sum_{k=0}^{n-1}\left(\sum_{j=0}^{k} \bar{a}_j\bar{b}_{k-j} + \sum_{j=k+1}^{n-1} \bar{a}_j\bar{b}_{k+n-j}\right) x^k \end{aligned}$$

Then, the n coefficients of the product polynomial are clear in this expression, so the NTT transform is as follows:

$$NTT_{\omega}(\Phi\left[a(x)b(x)\right]) = \sum_{i=0}^{n-1} \bar{C}_i x^i = \sum_{i=0}^{n-1}\left(\sum_{k=0}^{n-1}\left(\sum_{j=0}^{k} \bar{a}_j\bar{b}_{k-j} + \sum_{j=k+1}^{n-1} \bar{a}_j\bar{b}_{k+n-j}\right)(\omega^i)^k\right) x^i$$

On the other hand, we have the NTT transforms for $a(\phi x), b(\phi x)$

$$NTT_\omega(a(\phi x)) \cdot NTT_\omega(b(\phi x)) = \sum_{i=0}^{n-1} \bar{A}_i \bar{B}_i x^i = \sum_{i=0}^{n-1} \left(\sum_{k=0}^{n-1} \bar{a}_k (\omega^i)^k \sum_{j=0}^{n-1} \bar{b}_k (\omega^i)^k \right) x^i$$

Following the reasoning described for the product of the polynomials, we have:

$$\sum_{k=0}^{n-1} \bar{a}_k (\omega^i)^k \sum_{j=0}^{n-1} \bar{b}_k (\omega^i)^k = \sum_{k=0}^{n-1} \sum_{j=0}^{k} \bar{a}_j \bar{b}_{k-j} (\omega^i)^k + \omega^n \sum_{k=0}^{n-1} \sum_{j=k+1}^{n-1} \bar{a}_j \bar{b}_{k+n-j} (\omega^i)^k$$

$$NTT_\omega(a(\phi x)) \cdot NTT_\omega(b(\phi x)) = \sum_{i=0}^{n-1} \left(\sum_{k=0}^{n-1} \left(\sum_{j=0}^{k} \bar{a}_j \bar{b}_{k-j} + \sum_{j=k+1}^{n-1} \bar{a}_j \bar{b}_{k+n-j} \right) (\omega^i)^k \right) x^i$$

□

Theorem 3.4. *The result of the NTT multiplication is indeed the product $a(x)b(x)$.*

Proof. In the previous lemma, we have proven that given two polynomials $a(x)$, $b(x)$ the following equality holds:

$$NTT_\omega(\Phi\left[a(x)b(x)\right]) = NTT_\omega(a(\phi x)) \cdot NTT_\omega(b(\phi x))$$

Then, performing an inverse transformation on both sides:

$$\Phi\left[a(x)b(x)\right] = INTT_\omega(NTT_\omega(a(\phi x)) \cdot NTT_\omega(b(\phi x)))$$

The change of variables $\Psi : x \to \phi^{-1}x$ applied to the product polynomial yields the following result:

$$a(x)b(x) = \Psi \circ \Phi\left[a(x)b(x)\right] = \Psi\left[INTT_\omega(NTT_\omega(a(\phi x)) \cdot NTT_\omega(b(\phi x)))\right]$$

The formula on the right is the procedure described as the NTT multiplication. □

Chapter 4

Performance

In the following chapter we will present the performance of the reference implementations of Classic McEliece and NTRU for the NIST submission.

Definition 4.1. *Here we define the concepts presented in this chapter.*

- *The security levels relate to the hardness of a key search attack on a block cipher, like AES. That is, the comparison is made by taking into account the best known algorithms that break AES and the algorithms that break the analyzed KEMs. Specifically, NIST considers security levels 1, 3 and 5, which correspond to the security of AES-128, AES-192 and AES-256, respectively.*

- *The FPGA is the Hardware structure on which the cryptosystems have been implemented.*

- *The (clock) cycles of the algorithms are the amount of cycles it takes an algorithm to complete.*

- *The Frequency (in MHz) is the amount of millions Clock Cycles the computer is able to process per second. With higher frequency, an algorithm with the same number of clock cycles will take less real time in seconds to complete.*

- *The Lookup Tables (LUT) are hardware modules for performing simple operations. The number of LUTs in a hardware implementation corresponds to the amount of different logic operations that the algorithm will need to perform, regardless of how often they are used.*

- *Flip-flops (FF) measure the bits of memory the algorithm is using.*

4.1 Classic McEliece

Figure 4.1 shows Key and Ciphertext sizes for each parameter set. Figure 4.2 shows the performance and area of the Hardware implementations. The CPU on titan0 is an Intel Xeon E3-1275 v3 (Haswell) running at 3.50GHz. Some of the hardware designs for smaller parameter sets were built into an Artix-7 XC7A200T FPGA. In the remaining cases, the implementation is on a Virtex-7 XC7V2000T FPGA. [7]

For the highest security level implementation of Classic McEliece (largest parameters), mceliece8192128:

- The best performance achieved for the key generation process is 1286179 cycles in a Virtex-7 FPGA, although 4115427 cycles can be achieved with an approach better balanced with area.
- For the encapsulation, 6528 cycles are enough given that the implementation in Virtex-7 efficiently performs matrix multiplication.
- The best performance achieved for decapsulation is 26237 cycles in a Virtex-7 FPGA, and takes 33640 cycles in the balanced with area approach.

Performance decreases drastically when an Intel Haswell CPU core is used (not a specialized implementation like a FPGA).

- The key generation process takes 1277898472 cycles.
- Encapsulation takes 185146 cycles.
- Decapsulation takes 324803 cycles.

It is worth noting that the key generation process has a lot of variation (around 85%), because it does not always succeed, and is actually quite likely to not succeed on a given attempt.

	Sizes in bytes			
	Public key	Private key	Ciphertext	Session key
mceliece348864	261120	6452	128	32
mceliece460896	524160	13568	188	32
mceliece6688128	1044992	13892	240	32
mceliece6960119	1047319	13908	226	32
mceliece8192128	1357824	14080	240	32

Figure 4.1: Classic McEliece ciphertext and key sizes

	security	FPGA	keypair cycles	enc cycles	dec cycles	Fmax MHz	LUT	FF
Area optimized								
a	1	Artix-7	1599882	2720	15638	38.1	25327	49383
b	3	Artix-7	5002044	3360	27645	28.9	38669	74858
c	5	Virtex-7	12389742	5024	47309	37.3	44345	83637
d	5	Virtex-7	11179636	5413	40728	36.1	44154	88963
e	5	Virtex-7	15185314	6528	48802	33.4	45150	88154
Area and time balanced								
a	1	Artix-7	482893	2720	12036	30.4	39766	70453
b	3	Artix-7	1383104	3360	18771	23.5	57134	97056
c	5	Virtex-7	3346231	5024	32145	26.5	66615	111299
d	5	Virtex-7	3086064	5413	26617	30.7	63629	115580
e	5	Virtex-7	4115427	6528	33640	22.2	67457	115819
Time optimized								
a	1	Artix-7	202787	2720	10023	28.6	81339	132190
b	3	Virtex-7	515806	3360	14571	20.7	109484	168939
c	5	Virtex-7	1046139	5024	24730	16.0	122624	186194
d	5	Virtex-7	974306	5413	19722	28.8	116928	188324
e	5	Virtex-7	1286179	6528	26237	28.4	123361	190707

Figure 4.2: Performance and Area on the Classic McEliece reference implementation a=mceliece348864, b=mceliece460896, c=mceliece6688128 d=mceliece6960119, e=mceliece8192128

4.2 NTRU

Figure 4.3 shows Key and ciphertext sizes and cycle counts for all of the recommended parameter sets. Cycle counts were obtained on one core of an Intel Core i7-4770K (Haswell); "ref" refers to the C reference implementation, "AVX2" to the implementation using AVX2 vector instructions.[6]

Figure 4.4 shows the results of a Hardware implementation on the Xilinx Zynq UltraScale+ MPSoC XCZU9EG-2FFVB1156E[18]

ntruhrss701

Sizes (in Bytes)		Haswell Cycles (ref)		Haswell Cycles (AVX2)	
sk:	1,452	gen:	23,302,424	gen:	381,476
pk:	1,138	enc:	1,256,210	enc:	71,238
ct:	1,138	dec:	3,642,966	dec:	77,848

Figure 4.3: Performance and Sizes on the NTRU Reference implementation

Parameter Set	Security	Clock Freq. [MHz]	LUTs	FFs
ntruhps2048677	Level 1	200	24,328	19,244
ntruhps4096821	Level 3	200	29,389	23,338
ntruhrss701	Level 1	200	27,218	21,410

Figure 4.4: Area on the NTRU Reference implementation

4.3 Comparison

Here we present three graphics (Figures 4.5, 4.6, 4.7) comparing the metrics of both cryptosystems at security level 1. These graphics have been elaborated using the data from the previous tables.

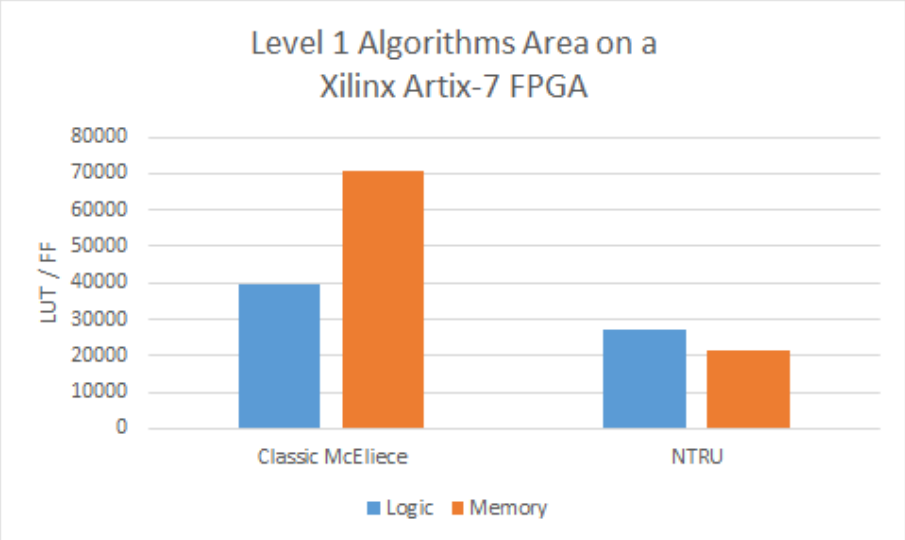

Figure 4.5: Logic and memory on the reference implementations

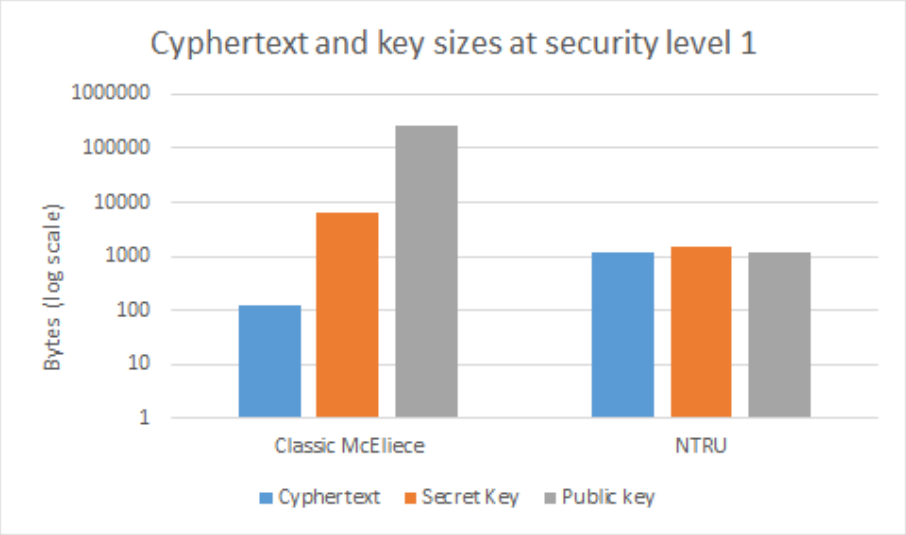

Figure 4.6: Ciphertext and key Sizes in logarithmic scale

Overall we can observe that the NTRU cryptosystem is more balanced across all metrics, while Classic McEliece makes compromises in some of them in order to shine in others.

The main aspect where NTRU rises as a clear winner over Classic McEliece is on the area used for their implementations. Classic McEliece requires about twice as much logic as NTRU, and more than three times the memory. In any context where the area is a major constraint, NTRU will be the best candidate amongst the two.

When we take a look at the ciphertext and key sizes, the discussion becomes a lot more nuanced. One may immediately discard Classic McEliece because of its large public key, of about 200 kB. This is the main weak point for this cryptosystem. Looking at the secret key, it is still bigger than NTRU, but by a more reasonable margin. For the first time in this comparison, we see that the Classic McEliece ciphertext is rather small, and it is indeed a whole order of magnitude smaller than NTRU. As we observed before, all sizes in NTRU are more balanced, all around 1kB.

The comparison of the performance is more favourable for Classic McEliece. The key generation is a bit slower than NTRU, but Encapsulation and Decapsulation are a lot faster.

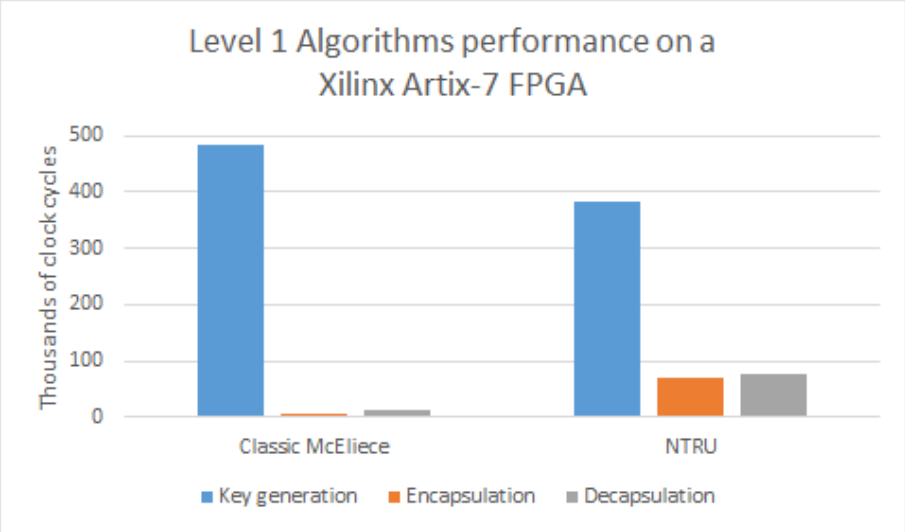

Figure 4.7: Algorithms Performance on the reference implementations

Given these numbers, one still may think that NTRU is a lot more favoured than Classic McEliece, and for the most part it is true. But to better grasp the nuance of this comparison, we need to take into account what these numbers mean and why we want them to be as small as possible. Regarding the area, we want to minimise it because when we use less area, we reduce both the required physical size of the implementation and the components needed, therefore reducing the overall cost.

When it comes to the sizes and the performance, we still want to minimize them, in order to save time and resources when storing and sharing the keys. The first thing to notice is that the speed of Key generation is a lot less relevant than the speed of Encapsulation and Decapsulation, because the keys are generated once, and then used several times for KEMS. Then, it is often considered that the performance of Key generation is not a relevant metric, because it is performed a lot less often that its counterparts. Something similar happens with the sizes of the keys. The public key, no matter how big it is, is something that only needs to be shared once. Also, since it is not secret information, it can be transmitted without any security protocols, which eases the process. The secret key, on the other side, is something that is not transmitted, but needs to be stored securely. Depending on the architecture of the implementation, needing to store large secret keys can be a downside. Again, the ciphertext is something that will be sent often, every time we encapsulate. In this sense, having a small ciphertext should be more relevant than having a large public key.

All of this comparisons show that neither cryptosystem has a clear advantage

over the other. Depending on the resources available to the users and the constraints on time, cost or size, different metrics will be more relevant than others, making it hard to establish a winner.

Bibliography

[1] E. R. Berlekamp. *Goppa codes*. IEEE Trans. Information Theory, IT-19(3):590–592, 1973.

[2] D. J. Bernstein, T. Chou and P. Schwabe. *McBits: Fast constant-time code-based cryptography.* Cryptographic Hardware and Embedded Systems – CHES 2013. Lecture Notes in Computer Science, pp. 250–272.

[3] Tung Chou. *McBits revisited.* In Wieland Fischer and Naofumi Homma, editors, Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, volume 10529 of Lecture Notes in Computer Science, pages 213-231. Springer, 2017.

[4] *Encryption Mechanisms Using a Software/Hardware Codesign Approach* URL: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[5] Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, RubenNiederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, JakubSzefer, Wen Wang, *Classic McEliece: conservative code-based cryptography, "Supporting Documentation"*(2019). URL: https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions

[6] Cong Chen, Oussama Danba, Jerey Hostein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, *NTRU: algorithm specifications and supporting documentation* (2019). URL: https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions

[7] Robert J. McEliece. *A public-key cryptosystem based on algebraic coding theory*. Technical report, NASA, 1978. URL: https://tmo.jpl.nasa.gov/progress_report2/42-44/44N.PDF

[8] James L. Massey, *Shift-register synthesis and BCH decoding*, IEEE Transactions on Information Theory 15 (1969), 122-127.

[9] Wen Wang, Jakub Szefer, and Ruben Niederhagen. *FPGA-based Niederreiter cryptosystem using binary Goppa codes.* In Tanja Lange and Rainer Steinwandt, editors, Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings, volume 10786 of Lecture Notes in Computer Science, pages 77-98. Springer, 2018.

[10] Andreas Hülsing, Joost Rijneveld, John Schanck, and Peter Schwabe. *High-speed key encapsulation from NTRU.* In Wieland Fischer and Naofumi Homma, editors, Cryptographic Hardware and Em- bedded Systems CHES 2017, LNCS. Springer, 2017. http://cryptojedi.org/papers/ntrukem

[11] Nick Howgrave-Graham, Joseph H. Silverman, Ari Singer, and William Whyte. *NAEP: Provable security in the presence of decryption failures.* Cryptology ePrint Archive, Report 2003/172, 2003. https://eprint.iacr.org/2003/172

[12] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *NTRU: A new high speed public key cryptosystem*, 1996. draft from at CRYPTO '96 rump session. http://web.securityinnovation.com/hubfs/files/ntru-orig.pdf

[13] Daniel J. Bernstein and Bo-Yin Yang. *Fast constant-time gcd computation and modular inversion.* Cryptology ePrint Archive, Report 2019/266, 2019. https://eprint.iacr.org/2019/266.

[14] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. Cheung, D. Pao, and I. Verbauwhede, *High-speed polynomial multiplication architecture for ring-lwe and she cryptosystems*, IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 62, no. 1, pp. 157-166, 2015.

[15] ETSI – European Telecommunications Standards Institute. *Quantum-Safe Cryptography (QSC); Quantum-safe algorithmic framework.* ETSI GR QSC 001 V1.1.1 (2016–07).

[16] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, Douglas Stebila.*Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange* Cryptology ePrint Archive, report 2018/903. 2018.

[17] Andreas Hülsing, Joost Rijneveld, John Schanck, and Peter Schwabe. *High-speed key encapsulation from NTRU*. In Wieland Fischer and Naofumi Homma, editors, Cryptographic Hardware and Embedded Systems CHES 2017, LNCS. Springer, 2017.

[18] F. Farahmand, V. Dang, M. Andrzejczak, K. Gaj. *Implementing and Benchmarking Seven Round 2 Lattice-Based Key Encapsulation Mechanisms Using a Software/Hardware Codesign Approach*. presented at the NIST Second PQC Standardization Conference, Santa Barbara, CA, USA, Aug. 22-24, 2019.

[19] P. Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM J. Comput., 26 (5), 1997, pp. 1484-1509.

[20] Jonathan Katz, Yehuda Lindell. *Introduction to Modern Cryptography* (Chapman & Hall/Crc Cryptography and Network Security Series), 2007.

[21] Oriol Farràs. *Quantum-resistant cryptography*. Technical report (2017).

[22] Daniel J.Bernstein, Johannes Buchmann, Erikand Dahmen. *Post Quantum Cryptography*, 2008, pp. 170-173.