Lattice-based cryptography - A comparative description and analysis of proposed schemes

Einar Løvhøiden Antonsen

Thesis submitted for the degree of
Master in Informatics: programming and networks
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2017

Lattice-based cryptography - A comparative description and analysis of proposed schemes

Einar Løvhøiden Antonsen

Acknowledgement

I would like to thank my supervisor, Leif Nilsen, for all the help and guidance during my work with this thesis. I would also like to thank all my friends and family for the support. And a shoutout to Bliss and the guys there (you know who you are) for providing a place to relax during stressful times.

Abstract

The standard public-key cryptosystems used today relies mathematical problems that require a lot of computing force to solve, so much that, with the right parameters, they are computationally unsolvable. But there are quantum algorithms that are able to solve these problems in much shorter time. These quantum algorithms have been known for many years, but have only been a problem in theory because of the lack of quantum computers. But with recent development in the building of quantum computers, the cryptographic world is looking for quantum-resistant replacements for today's standard public-key cryptosystems.

Public-key cryptosystems based on lattices are possible replacements. This thesis presents several possible candidates for new standard public-key cryptosystems, mainly NTRU and ring-LWE-based systems. The lattice-based cryptosystems are shown to be very fast and have strong, provable security against quantum computers, but are a lot more complicated than RSA and Diffie-Hellman.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Keeping information hidden and secret has been a part of human history for a long time. A lot of different techniques have been devised and used, from simple substitution ciphers through the Enigma machine to today's standarised digital algorithms. And along with the development of cryptographic systems, comes the development of attacks against these systems. This is one of the main driving forces for the development of new cryptographic systems, and the main reason NSA, in 2015, announced that their Suite B will be transitioning to new cryptographic algorithms in the near future [2]. The reason for this announcement is that there exists quantum algorithms (algorithms running on a quantum computer) that can break some of the most used cryptographic systems. These algorithms have been known since the 1990's, but have not been a true threat because of the lack of quantum computers. Not until now, it seems. The NSA states that it is likely a quantum computer capable of breaking public key cryptography will be constructed within a few decades [26].

Even though the announcement from the NSA came in 2015, work on alternative, quantum safe cryptographic systems have been ongoing since the quantum algorithms were discovered. Today, there are several schemes that are possible replacements of the current standard schemes. These new schemes can be divided into different categories such as hash-based, lattice-based, and code-based schemes. This thesis will focus on lattice-based cryptographic schemes.

1.1 Methodology and structure

The intention with this theses is to describe lattice-based cryptographic systems and do a comparative description and analysis of some of the proposed systems. The comparisons will be based on aspects of the cryptographic schemes such as key sizes and run times. The proposed schemes will also be compared to the standard systems used today, such as Diffie-Hellman and RSA. Also, by describing and comparing lattice-based cryptosystems, the thesis aims to make lattice-based cryptography a bit more accessible, as a lot of the information on it is "hidden" in technical papers with quite advanced mathematics.

The thesis will first, in chapter 2, describe some of the standard cryptographic systems used today, and describe how quantum computers are a threat to these systems. Chapter 3 will then present mathematical background of lattices and computational problems in lattice theory. Chapter 4 describes the NTRU cryptosystem by first presenting the protocol of the system, then describes the security of the system, and lastly describes the performance. Chapter 5 describes Chris Peikert's cryptosystem based on the ring-LWE problem. Chapter 6 describes two implementations of the cryptosystem presented by Peikert. First *Post-Quantum Key Exchange for TLS*, then *NewHope*. Chapter 7 describes two other implementations of ring-LWE cryptosystems, namely Singh's *Practical Key Exchange for the Internet* and *Even More Practical Key Exchange for the Internet.* Chapter 8 compares the presented cryptosystems, and also some standard systems used today. Chapter 9 is a small summary of the thesis, and also presents possible further work.

Chapter 2

Cryptographic Background

To be able to hide information from other people have been a part of human history for a long time. Using codes and ciphers began thousands of years ago. As most other technologies, cryptography have changed quite a lot the last hundred years. This chapter will describe some of the standard cryptographic systems we use today and why quantum computers are a threat to them.

2.1 Asymmetric key algorithms

Up until the 1970's, encryption and decryption had required that all parties needed to know a secret key. This secret key was used for both encryption and decryption. But in 1976, Whitfield Diffie and Martin Hellman released a paper [7] that would change the world of cryptography forever. The paper described what is known as public-key cryptography. The idea is that two parties can encrypt and decrypt data without sharing a secret key. The paper also described a cryptographic system allowing two parties to agree on a secret key without sharing the actual key with each other. This cryptographic system became know as Diffie-Hellman key exchange, and started the research and development of a new type of cryptography, the asymmetric key algorithms. These algorithms can be grouped in three different categories:

1. asymmetric cryptosystems,
2. digital signatures, and
3. key agreement systems.

One property that asymmetric key algorithms have in common is that they base their security on mathematical properties that are computationally hard to reverse, also called "hard problems". Integer factorization and discrete logarithm are examples on such problems. We will take a closer look on these problems when describing the three categories of asymmetric key algorithms.

2.1.1 Asymmetric cryptosystems

Encrypting and decrypting are two essential operations in cryptography. As mentioned earlier, this used to require a secret shared between the parties. After the public discovery of the Diffie-Hellman key exchange, the hunt for an asymmetric cryptosystem for encrypting and decrypting data started. And in 1977, Ron Rivest, Adi Shamir, and Leonard Adleman released a paper decribing such a system [24]. The cryptosystem is known as RSA, after the initials of their surnames.

So how is an asymmetric cryptosystem constructed? Instead of a shared secret key used for both encryption and decryption, you have a key-pair: one key for encryption and one key for decryption. The encryption key E is usually in the public domain (public key), while the decryption key D is kept private (private key). Asymmetric cryptosystems are therefore also called Public-key cryptosystems. The keys are created such that any data encrypted with E can only be decrypted with the corresponding D.

Using the keys is very simple. If Alice wants to send an encrypted message M to Bob, she gets Bob's public key E_B, encrypts the message, and sends the ciphertext C. Bob then use his private key D_B to decrypt C, and gets M as a result. Because of how the keys are constructed, Bob is the only one who can decrypt C, given he has kept his decryption key private. So an asymmetric cryptosystems consists of the following operations:

- Key generation
- Encryption
- Decryption

As mentioned earlier, asymmetric cryptosystems are based on "hard problems". In RSA, the security is based on two such problems: the problem of factoring large numbers and the RSA problem. The problem of factoring large number is simply that there are no known efficient algorithms for factoring large numbers, so it is not possible to factorize large numbers in a reasonable time. The RSA problem refers to how the RSA system is constructed and is defined as follows: recovering a value m such that $c \equiv m^e \pmod{n}$, where (n, e) is an RSA public key and c is an RSA ciphertext. It is believed that factoring n is the best approach to solve the RSA problem, so as long as integer factorization remains a hard problem, the RSA problem will also be a hard problem.

2.1.2 Digital signatures

While encryption of data typically ensures that the data cannot be accessed by unauthorized entities, it does not necessarily provide other security aspects such as integrity and non-repudiation. This is especially true with asymmetric cryptosystems. Since encryption is done with a public key, you don't get any information about the sender, unless they provide it. And this is where digital signatures come into the picture. Asymmetric cryptography can be used to create a digital signature scheme. You need three different algorithms:

- A key generation algorithm, creating a private and public key pair.
- A signing algorithm.
- A verification algorithm.

A digital signature scheme is very similar to a asymmetric cryptosystem. Rivest, Shamir, and Adleman describes in [24] how their cryptosystem also can be used for digital signatures. If Bob wants to send a signed message M to Alice, he uses his private (decryption) key D_B on M and computes his signature S. He then uses Alice's public (encryption) key E_A on S and sends the resulting ciphertext C to Alice. Alice can then use her private key D_A on C to retrieve S, and then use Bob's public key E_B on S to get the message M. By using the keys like this, ensures that Alice can be sure that it was Bob that sent the message.

The security of digital signature schemes are based on the same problems as asymmetric cryptosystems.

2.1.3 Key agreement systems

In some cases, it is preferable to use symmetric cryptosystems for encrypting data. Symmetric cryptosystems are usually faster than asymmetric cryptosystems, so they are better if you are encrypting a lot of data. However, they are using the same key for both encryption and decryption, both parties of the transmission must know the secret key. You can't just send the key to each other, so you need a system to agreeing on a key. One method is for one person to generate a key and then use an asymmetric cryptosystem to encrypt it and send to the recipient. This is not really a key agreement system, as key agreement systems require that all parties influence the creation of the key, but rather what is called key transport. Here, only one person has any influence on the key creation, so this is actually called a key transport.

The Diffie-Hellman key exchange is a key agreement system. It is constructed similarily to asymmetric cryptosystems, and works as follows:

1. Alice and Bob agrees on a modulus p and base g.
2. Alice chooses a secret integer a, and sends Bob $A = g^a \pmod{p}$.
3. Bob chooses a secret integer b, and sends Alice $B = g^b \pmod{p}$.
4. Alice computes $s = B^a \pmod{p}$.
5. Bob computes $s = A^b \pmod{p}$.
6. Alice and Bob now shares the secret s.

This works because $A^b \mod p = g^{ab} \mod p = g^{ba} \mod p = B^a \mod p$.

Alice and Bob have now created a shared secret key, and all parties influenced the creation. This key agreement system base its security on the discrete logarithm problem, which state that it is computationally hard to compute the integer k exponent solving the equation $b^k = g$, where b and g are elements of a finite group.

2.1.4 Cryptographic definitions

In chapter 5, some definitions of cryptosystems from Chris Peikert's paper on ring-LWE [22] will be used. The definitions are presented here.

Public-key cryptosystem

A public-key cryptosystem (PKC) consists of a ciphertext space C and a message space M and is given by four efficient algorithms:

- Setup() outputs a public parameter pp.
- Gen(pp) outputs a public encryption key pk and a secret decryption key sk.
- Enc(pp, pk, μ) takes a public key pk and a message $\mu \in M$, and outputs a ciphertext $c \in C$.
- Dec(sk, c) takes a decryption key sk and a ciphertext c, and outputs some $\mu \in M \cup \{\perp\}$, where $\perp$ is some distinguished symbol denoting decryption failure.

Key encapsulation mechanism

A key encapsulation mechanism (KEM) is a one-message protocol for transmitting a secret key to a receiver, using the receiver's public key. A KEM consists of ciphertext space C and key space K and is given by four efficient algorithms:

- Setup() outputs a public parameter pp.
- Gen(pp) takes the public parameter and outputs a public encapsulation key pk and a secret decapsulation key sk.
- Encaps(pp, pk) takes the public parameter and an encapsulation key pk, and outputs a ciphertext $c \in C$ and a key $k \in K$.
- Decaps(sk, c) takes a decapsulation key sk and a ciphertext c, and outputs some $k \in K \cup \{\perp\}$, where $\perp$ is some distinguished symbol denoting decapsulation failure.

A KEM is passively secure if the key k outputted from Encaps(pp, pk) is computationally indistinguishable from a random key $k^* \in K$. This is more formally described as satisfying IND-CPA security, where the outputs of the following games are indistinguishable:

$$\begin{aligned}
&pp \leftarrow \text{Setup}()\\
&(pk, sk) \leftarrow \text{Gen}(pp)\\
&(c, k) \leftarrow \text{Encaps}(pp, pk)\\
&\text{Output } (pp, pk, c, k)
\end{aligned}$$

and

$$\begin{aligned}
&pp \leftarrow \text{Setup}()\\
&(pk, sk) \leftarrow \text{Gen}(pp)\\
&(c, k) \leftarrow \text{Encaps}(pp, pk)\\
&k* \leftarrow K\\
&\text{Output } (pp, pk, c, k*)
\end{aligned}$$

The first game is called the "real" game and the second is called the "ideal" game. This form of security is also called indistinguishability under chosen-plaintext attack, and requires that the attacker is not able to decrypt any encrypted messages.

If the attacker is able to decrypt chosen messages, it is called indistinguishability under chosen ciphertext attack (IND-CCA). If the two games still are indistinguishable, the KEM is actively secure.

2.2 Quantum computers

Quantum computers are computers using quantum mechanics to operate. This allows a quantum computer to behave different than a classical computer, and solve problems that were thought to be computationally unsolvable. Problems that Diffie-Hellman and RSA base their security on.

2.2.1 Qubits

Ordinary computers are using bits to perform computations. These bits are either 0 or 1, nothing more nothing less. At any given point in time, the state of a computer can be described by a single string of these bits [11]. A quantum computer, on the other hand, incorporates quantum mechanics in how computations are performed. Instead of operating on bits, a quantum computer uses qubits (quantum bits). Qubits can still be 0 or 1, but can also hold these two values at the same time. This phenomenom is known as a superposition of two states. It is not possible to observe the superposition, as once you try to measure the state of a qubit it falls into either 0 or 1, decided by some probability [20].

Since the qubits are following the laws of quantum mechanics, it makes it possible to have quantum computers behave very different than ordinary computers. For instance, preparing a string of qubits of the same length in the same way, does not always result in the same bit string [16].

2.2.2 Shor's algorithm

A lot of cryptographic systems base their security on computationally hard problems, meaning that the current best algorithms for solving the problems are not efficient enough to create practical attacks. One of these problems is integer factorisation (given an integer $N = p \times q$ for some prime numbers p and q, determine p and q). The current fastest algorithm for factoring large numbers on a classical computer is the general number field sieve, and runs in

$$O(exp(\frac{64}{9}n^{1/3}(\log n)^{2/3}))$$

operations, where n is the number of bits used to represent the number [12]. But a much faster quantum algorithm was presented already in 1994 by Peter Shor [27]. Shor's algorithm runs in

$$O((\log n)^2 \cdot \log\log n)$$

on a quantum computer [12], and utilizes quantum parallelism. Since qubits can be in a superposition, quantum memory holds in theory every value until it is measured. So instead of doing an operation on each number, you can do the operation once. This is where the speedup lies. The algorithm does not always find the correct factor, as it relies on some probability, but the speedup compared to classical algorithms is still very high even if you have to run the algorithm several times.

2.2.3 Grover's algorithm

A very common problem in computer science is to search for an element in an unsorted set that satisfy one or more conditions. This is called the unstructured search problem. To solve this problem with certainty, any algorithm must do $N = 2^n$ evaluations in worst case, where n is the number of binary variables in the set [19]. But in 1996 [10], a quantum algorithm was presented by Lov K. Grover which solves this problem using $O(\sqrt{N})$ in the worst case. Similar to Shor's algorithm, Grover's algorithm utilizes the superposition of the qubits to achieve the speedup.

One really interesting aspect of Grover's algorithm is that it can be applied to any problem in the complexity class NP [19]. By applying Grover's algorithm to an efficient classical checking algorithm for certificates, and searching over all possible certificates, we get a speedup from $O(2^m poly(m))$ to $O(2^{m/2} poly(m))$. This is nearly a quadratic speedup, and implies that a problem instance of approximately twice the size can be solved in the same time on a quantum computer with the same clock speed compared to a normal computer.

Chapter 3

Lattices

To get an understanding of lattice-based cryptography, we need to know the underlying mathematics. The term lattice is used in both group theory and order theory. In lattice-based cryptography, we are using lattices as defined in group theory. In this chapter, we will describe the basic theory of lattices, how they are defined, and what problems in lattice theory that are usable in cryptographic systems. Most of the material in this chapter is taken from the notes of Chi et al. [6], Peikert's paper on lattice cryptography [21], and Micciancio and Regev's chapter about lattice-based cryptography [18].

3.1 Basic Definitions

Lattice

A very short definition of a lattice is that a lattice L of $\mathbb{R}^n$ is a discrete subgroup of $\mathbb{R}^n$. One can also say that a lattice is a set of points in n-dimensional space with a periodic structure. We will only consider integer lattices, i.e, $L \subset \mathbb{Z}^n$.

An n-dimensional lattice L must satisfy two additional properties:

1. L is an additive subgroup, which means that for every $x, y \in L$, we have $0 \in L$ and $-x, x + y \in L$.

2. L is discrete, which means that every $x \in L$ has a neighbourhood in $\mathbb{R}^n$ in which x is the only lattice point.

An example is, for instance, the integer lattice $\mathbb{Z}^n$.

Bases

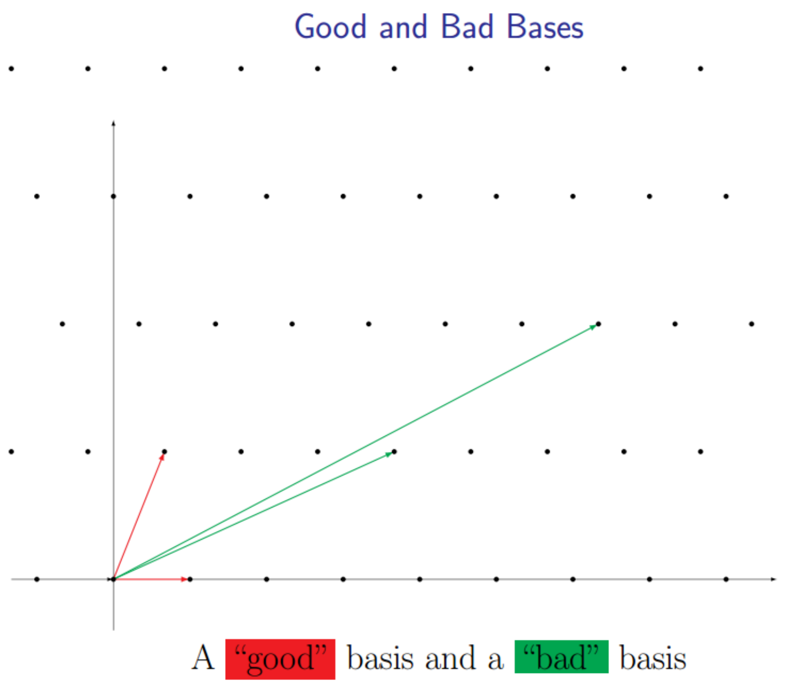
Lattices are generated from some basic vectors, which are linearly independent and often denoted as $\mathbb{B} = (b_1, b_2, \ldots, b_n)$, where n is the dimension of the lattice.

Definition 3.1.1. *A basis of L is an ordered set $\mathbb{B} = (b_1, b_2, \ldots, b_n)$ such that*

$$L = L(\mathbb{B}) = \mathbb{B} \cdot \mathbb{Z}^n = \left\{ \sum_{i=1}^{n} c_i b_i \ : \ c_i \in \mathbb{Z} \right\}. \tag{3.1}$$

Bases can be "good" or "bad", affecting how the resulting lattice ends up. A basis is good if the vectors are pairwise reasonably orthogonal, meaning they should pairwise make an angle of 90° or close to it [28].

Figure 3.1: Example of a good and a bad basis [28].



Definition 3.1.2. *The fundamental parallelepiped of basis* $\mathbb{B}$ *is*

$$\begin{aligned} P(\mathbb{B}) &= \mathbb{B} \cdot \left[-\frac{1}{2}, \frac{1}{2} \right)^n \\ &= \left\{ \sum_{i=1}^{n} \alpha_i b_i \ : \ -\frac{1}{2} \leq \alpha_i < \frac{1}{2} \right\}. \end{aligned} \tag{3.2}$$

As mentioned, a good basis gives a square-like parallelepiped with angles close to 90°, while a bad basis gives a very thin parallelepiped. See figure 3.1 for an example. This gives us the following lemma.

Lemma 3.1.1.

$$\mathbb{R}^n = \bigcup_{v \in L} (v + P(\mathbb{B})), \tag{3.3}$$

that is, parallel translation by lattice vectors of parallelepiped covers $\mathbb{R}^n$ *without overlap.*

Proof. For any $p \in \mathbb{R}^n$,

$$\begin{aligned} p &= \sum_i x_i b_i \\ &= \sum_i \lceil x_i \rfloor b_i + \sum (x_i - \lceil x_i \rfloor) b_i, \end{aligned} \tag{3.4}$$

where $\lceil a \rfloor$ means rounding off. Therefore,

$$-\frac{1}{2} \leq a - \lceil a \rfloor < \frac{1}{2}. \tag{3.5}$$

Hence, $\sum_i \lceil x_i \rfloor b_1 \in L$ and $\sum_i (x_i - \lceil x_i \rfloor) b_i \in P(\mathbb{R})$. This shows that $\mathbb{R}^n = \bigcup_{v \in L} (v + P(\mathbb{B}))$.

If $(v_1 + P(\mathbb{B})) \cap (v_2 + P(\mathbb{B})) \neq \emptyset$ for some $v_1 \neq v_2 \in L$, then $v_1 + \alpha = v_2 + \beta$ for some $\alpha, \beta \in P(\mathbb{B})$, so $v_1 - v_2 = \beta - \alpha$. Since $v_1 - v_2$ is a $\mathbb{Z}$-linear combination of b_i while $\beta - \alpha$ is a $(-1, 1)$-linear combination of b_i, so $v_1 - v_2 = 0 = \beta - \alpha$. □

Coset and Determinant

It is possible to think of a coset element of $\mathbb{Z}^n / L$ as a subset $v + L$, i.e., a shift of the lattice L, where $v \in \mathbb{Z}^n$ represents a coset of $\mathbb{Z}^n / L$.

Lemma 3.1.2. *Each coset of L has a unique representative in a parallelepiped $P(\mathbb{B})$, because $\bigcup_{v\in L}(v+P(\mathbb{B}))$ covers $\mathbb{R}^n$ without overlap.*

Proof. Let $v \in \mathbb{Z}^n$ be a representative of a coset $v+L$. Since $\bigcup_{v\in L}(v+P(\mathbb{B}))$ covers $\mathbb{R}^n$ without any overlap, there exists a unique $w \in L$ such that $v \in (W+P(\mathbb{B}))$. Then $v-w \in P(\mathbb{B})$, and v represents the same coset, i.e.,

$$v+L=(v-W)+L, \tag{3.6}$$

so $v-w$ is a representative of the coset $v+L$ in $P(\mathbb{B})$. Moreover, such a representative is unique, since if $v_1, v_2 \in P(\mathbb{B})$ and

$$v_1+L=v_2+L, \tag{3.7}$$

where

$$\begin{aligned} v_1 &= \sum c_{1j}b_j, \qquad -\frac{1}{2} \leq c_{1j} < \frac{1}{2}, \\ v_2 &= \sum c_{2j}b_j, \qquad -\frac{1}{2} \leq c_{2j} < \frac{1}{2} \end{aligned} \tag{3.8}$$

then

$$v_1-v_2=\sum(c_{1j}-c_{2j})b_j \in L, \tag{3.9}$$

i.e., $c_{1j}-c_{2j} \in \mathbb{Z}$ for all j. Note that if $-\frac{1}{2} \leq a \leq \frac{1}{2}$ and $-\frac{1}{2} \leq b \leq \frac{1}{2}$, then $-1 \lneqq a-b \lneqq 1$. Hence, $c_{1j}-c_{2j}=0$ for $j=1,2,\ldots,n$. □

Definition 3.1.3. *The determinant of a lattice L is defined as*

$$det(L) := |\mathbb{Z}^n/L| = |det(\mathbb{B})| = vol(P(\mathbb{B})) \tag{3.10}$$

for any basis $\mathbb{B}$ of L.

Lemma 3.1.3. $|\mathbb{Z}^n/L| = vol(P(\mathbb{B}))$.

Proof. Note the following:

- $L+P(\mathbb{B})$ covers $\mathbb{R}^n$ without overlap.
- $\mathbb{Z}^n + \boxdot$ covers $\mathbb{R}^n$ wihtout overlap, where $\boxdot$ means the half closed unit cube $\left[-\frac{1}{2}, \frac{1}{2}\right)^n$.

Thus,

$$\begin{aligned} L+P(\mathbb{B}) &= \mathbb{R}^n \\ &= \mathbb{Z}^n + \boxdot \\ &= \bigcup_{c\in\mathbb{Z}^n/L}(c+L+\boxdot). \end{aligned} \tag{3.11}$$

It follows that $|\mathbb{Z}^n/L||\boxdot| = |P(\mathbb{B}|$, so $|\mathbb{Z}^n/L| = vol(P(\mathbb{B}))$. □

Successive Minima

Definition 3.1.4. *Successive minima if linearly independent vectors are defined by the following two properties:*

- $\lambda_1(L) := min_{0 \neq v \in L} \|v\| = min_{x \neq y \in L} \|x - y\|$
- $\lambda_i(L) := min\{r : L \text{ contains } i \text{ linearly independent vectors of length } \leq r\}$.

We then have that $\lambda_1(L) \leq \lambda_2(L) \leq \ldots \leq \lambda_n(L)$. Let $v_1, v_2, \ldots, v_n$ be corresponding lattice elements. $\{v_1, v_2, \ldots, v_n\}$ does not need to be a basis of L. $\|v\|$ is the Euclidian norm of v.

Q-ary lattices

A special kind of lattices are usually used in lattice-based cryptographic systems, namely *q-ary lattices.* Q-ary lattices have the additional property that a lattice L satisfy $q\mathbb{Z}^n \subseteq L \subseteq \mathbb{Z}^n$ for some (possibly prime) integer q [18]. This means that, for a vector $x \in \mathbb{Z}^n$, x is in L if and only if $x \mod q$ also is in L.

Gram-Schmidt Orthogonalization and Lower Bounding λ_1

The Gram-Schmidt orthogonalization $\tilde{\mathbb{B}}$ of a basis $\mathbb{B}$ of L is given by

$$\begin{aligned} \mathbb{B} &= QR \\ &= Q \begin{pmatrix} \|\tilde{b_1}\| & & * \\ & \ddots & \\ 0 & & \|\tilde{b_n}\| \end{pmatrix} \\ &= \tilde{\mathbb{B}} \begin{pmatrix} 1 & & * \\ & \ddots & \\ 0 & & 1 \end{pmatrix}, \end{aligned} \tag{3.12}$$

where

$$\tilde{\mathbb{B}} = Q \begin{pmatrix} \|\tilde{b_1}\| & & 0 \\ & \ddots & \\ 0 & & \|\tilde{b_n}\| \end{pmatrix}, \tag{3.13}$$

and Q is an orthonormal basis reduced from $\tilde{\mathbb{B}}$, and R is a representation of $\mathbb{B}$ with respect to this basis.

Lemma 3.1.4. *$P(\tilde{\mathbb{B}} = \tilde{\mathbb{B}}\dot{[} - \frac{1}{2}, \frac{1}{2})^n$ is a fundamental domain of L. That is, $L + P(\tilde{\mathbb{B}})$ covers $\mathbb{R}^n$ without overlap.*

Proof. Since $vol(P(\tilde{\mathbb{B}})) = vol(P(\mathbb{B}))$, it sufficies to show that there is no overlap. Assume there is an overlap, i.e.,

$$\mathbb{B}x + \tilde{\mathbb{B}}\alpha = \mathbb{B}y + \tilde{\mathbb{B}}\beta \tag{3.14}$$

for some $x, y \in \mathbb{Z}^n$ and $\vec{\alpha}, \vec{\beta} \in [-\frac{1}{2}, \frac{1}{2})^n$. Then $\mathbb{B}(x - y) = \tilde{\mathbb{B}}(\vec{\beta} - \vec{\alpha})$. Letting $z = x - y$,

$$\tilde{\mathbb{B}} \begin{pmatrix} 1 & & * \\ & \ddots & \\ 0 & & 1 \end{pmatrix} z = \tilde{\mathbb{B}}(\vec{\beta} - \vec{\alpha}), \tag{3.15}$$

so

$$\begin{pmatrix} 1 & & * \\ & \ddots & \\ 0 & & 1 \end{pmatrix} z = (\vec{\beta} - \vec{\alpha}). \tag{3.16}$$

Note that z is an integer vector and

$$-1 \lneqq \beta_i - \alpha_i \lneqq 1. \tag{3.17}$$

From the equality 3.16

$$\begin{aligned} z_n &= \beta_n - \alpha_n \quad \therefore z_n = 0 \to \alpha_n = \beta_n \\ z_{n-1} + *z_n &= \beta_{n-1} - \alpha_{n-1} \\ z_{n-1} &= \beta_{n-1} - \alpha_{n-1} \quad \therefore z_{n-1} = 0 \to \alpha_{n-1} = \beta_{n-1} \\ &\cdots \\ \therefore z_1 &= 0 \\ i.e., \ x &= y. \end{aligned} \tag{3.18}$$

□

Minkowski's Theorem and Upper Bounding λ_1

Theorem 3.1.1. *Minkowski's Theorem 1: Any convex centrally symmetric body S of volume greater than 2^n det(L) contains a nonzero lattice point.*

Proof. Let $S' = \frac{1}{2}S$, so $vol(S') > det(L)$. Then there exist $x \neq y \in S'$ such that $x - y \in L$, since for some $v_1 \neq v_2 \in L$,

$$\begin{aligned}
&(v_1 + S') \bigcap (v_2 + S') \neq \phi \\
&z = v_1 + x = v_2 + y,\ x, y \in S' \qquad (3.19)\\
&x - y = v_2 - v_1 \neq 0 \in L.
\end{aligned}$$

Now $2x, -2y \in S$ by the definition of S', so

$$x - y = \frac{1}{2}(2x) + \frac{1}{2}(-2y) \in S \qquad (3.20)$$

by the convexity of S. □

Corollary 3.1.1.

$$\lambda_1(L) \leq \sqrt{n}(det(L))^{\frac{1}{n}}. \qquad (3.21)$$

Proof. The corollary is proven by using the following two facts:

- A ball of radius $> \sqrt{n}(det(L))^{\frac{1}{n}}$ is convex and centrally symmetric.
- $B(0, \sqrt{n}(det(L))^{\frac{1}{n}}) \supset$ a cube of side length $2(det(L))^{\frac{1}{n}}$, since

$$dist((1, \ldots, 1), (0, \ldots, 0)) = \sqrt{n}.$$

It follows that

$$vol(B(0, \sqrt{n}(det(L))^{\frac{1}{n}})) > 2^n det(L).$$

□

Remark 3.1.1. *A more refined inequality could be obtained if the exact formula for $vol(B(0,R))$ is used. Choose R such that $vol(B(0,R)) = 2^n det(L)$. Then $\lambda_1(L) \leq R$ [6].*

Theorem 3.1.2. *Minkowski's Theorem 2:* $(\prod_{i=1}^{n} \lambda_i(L))^{\frac{1}{n}} \leq \sqrt{n}(det(L))^{\frac{1}{n}}$.

Proof. We may assume $\|b_i\| = \lambda_i(L)$ for $i = 1, \ldots, n$, and consider a lattice generated by $b_1, \ldots, b_n$, possibly a sublattice of L.

$$T := \left\{ y \in \mathbb{R}^n \ :\ \sum_{i=1}^{n} \left(\frac{\langle y, \tilde{b_i} \rangle}{\|\tilde{b_i}\| \lambda_i} \right)^2 < 1 \right\}. \qquad (3.22)$$

Claim: the ellipsoid T does not contain any nonzero lattice point. Let $0 \neq y \in L$, and $1 \leq k \leq n$ maximal such that

$$\lambda_{k+1}(L) \geqq \|y\| \geq \lambda_k(L). \tag{3.23}$$

The claim is $y \in span\{b_1, \ldots, b_k\} = span\{\tilde{b_1}, \ldots, \tilde{b_k}\}$. If not, $b_1, \ldots, b_k, y$ are $k+1$ linearly independent and their norms are less than λ_{k+1}, a contradiction. Hence,

$$\begin{aligned}\sum_{i=1}^{n} \left(\frac{\langle y, \tilde{b_i} \rangle}{\|\tilde{b_i}\| \lambda_i} \right)^2 &= \sum_{i=1}^{k} \left(\frac{\langle y, \tilde{b_i} \rangle}{\|\tilde{b_i}\| \lambda_i} \right)^2 \\ &\geq \sum_{i=1}^{k} \frac{1}{\lambda_k^2} \left(\frac{\langle y, \tilde{b_i} \rangle}{\|\tilde{b_i}\|} \right)^2 \\ &= \frac{\|y\|^2}{\lambda_k^2} \geq 1,\end{aligned} \tag{3.24}$$

so $y \notin T$, i.e., T does not contain any nonzero lattice vector. Hence,

$$2^n det(L) \geq vol(T) = \left(\prod_{i=1}^{n} \lambda_i \right) vol(B(0:1)) \geq \left(\prod_{i=1}^{n} \lambda_i \right) \left(\frac{2}{\sqrt{n}} \right)^n, \tag{3.25}$$

so

$$\left(\prod_{i=1}^{n} \lambda_i \right)^{\frac{1}{n}} \leq \sqrt{n} (det(L))^{\frac{1}{n}}. \tag{3.26}$$

□

3.2 Computational problems

In the previous chapter, it was described how asymmetric cryptographic systems base their security on computational hard problems. To be able to construct such cryptosystems based on lattices, there have to exist similarily hard problems in lattice theory. In fact, there are several potential hard problems, but the most well known are:

- Shortest Vector Problem (SVP).
- Closest Vector Problem (CVP).

- Shortest Independent Vectors Problem (SIVP).

The base versions of these problems are usually not used in lattice-based cryptography, but rather the approximation versions of them are used [18]. These approximation versions are denoted by adding the subscript γ, so that instead of SVP, you have SVP$_\gamma$. We will describe some of these problems in more detail in the following sections.

3.2.1 Shortest Vector Problem

The shortest vector problem is one of the most well studied problems in lattice theory, and is defined as follows: Given an arbitrary basis B of some lattice $L = L(B)$, find the shortest non-zero lattice vector, i.e. $v \in L$ for which $\|v\| = \lambda_1(L)$. $\lambda_1(L)$ denotes the minimum distance of the lattice L, which is the length of the shortest non-zero lattice vector [21]. As mentioned, this problem is usually used in the approximation version, and is then defined as follows: Given a basis B of an n-dimensional lattice $L = L(B)$, find a non-zero vector $v \in L$ for which $\|v\| \leq \gamma(n) \times \lambda_1(L)$. The approximation factor γ is larger or equal to 1 and is usually taken to be a function of the lattice dimension n, i.e $\gamma = \gamma(n)$. By setting $\gamma = 1$, SVP$_\gamma$ is equal to SVP.

3.2.2 Closest Vector Problem

The closest vector problem is a bit similar to SVP. It is defined as follows: Given a lattice basis B and a target vector t (not necessarily in the lattice), find the lattice point $v \in L(B)$ closest to t [18]. So instead of finding the shortest vector in the whole lattice, one must find the shortest vector from a given point in the lattice. The approximation problem CVP$_\gamma$ require you to find a vector whose norm is at most γ times the shortest vector, similar to SVP$_\gamma$. CVP is actually a generalisation of SVP. It is possible to reduce SVP$_\gamma$ to CVP$_\gamma$ [9].

3.2.3 Shortest Independent Vectors Problem

The shortest independent vectors problem is another computational problem in lattice theory. The problem is defined as follows: Given a lattice basis $B \in \mathbb{Z}^{n \times n}$, find n linearly independent lattice vectors $S = [s_1, \ldots, s_n]$ (where $s_i \in L(B)$ for all i) minimizing the quantity $\|S\| = \max_i \|s_i\|$ [18].

3.2.4 Complexity

The reason these problems in lattice theory are used in cryptographic systems, is that they are computationally hard to solve. One of the best polynomial-time algorithms for solving these problems is the LLL-algorithm, presented by Lenstra, Lenstra, and Lovász in 1982. The LLL-algorithm only yields slightly subexponential approximation factors $\gamma = 2^{\Theta(n \log \log n / \log n)}$. Algorithms that give polynomial or better approximation factors, all require super-exponential $2^{\Theta(n \log n)}$ time, or exponential $2^{\Theta(n)}$ and space [21]. These algorithms are therefore not possible to use to solve the problems, as they require either too much time or space. Many lattice problems are actually known to be NP-hard. But such hardness does not really effect cryptography, since lattice-based cryptosystems so far rely on polynomial approximation problems factors $\gamma(n) \geq n$ [21].

Chapter 4

NTRU

One of the most well known lattice-based cryptographic systems, is NTRU. It was proposed by Hoffstein, Pipher, and Silverman in 1996, and is a ring-based public key cryptosystem [15]. The cryptosystem consists of two algorithms: NTRUEncrypt and NTRUSign. As the names implies, they are used for encryption and digital signatures, respectively. NTRU does not have a key exchange protocol. The cryptosystem was patented by the developers in 1996, but in 2013 they released the intellectual property and a sample implementation under the Gnu Public License, hoping for a more widespread adoption of the system. An updated paper was released in 1999 for submission of the NTRU public key cryptosystem for consideration for inclusion into the P1363A standard [14].

The NTRU system uses a mixing system. The encryption procedure is based on polynomial algebra and reduction modulo two numbers p and q. The decryption procedure uses an unmixing system which depends on elementary probability theory. The security of the system is based on the interaction of the polynomial mixing system with the independence of reduction modulo p and q, as well as that it is very difficult to find extremely short vectors in most lattices [15].

In this chapter, we will go into the details of the NTRU cryptosystem. First, we will describe the protocol and the theory of the system. Then we will describe the performance of the system, such as key sizes and running times. Lastly, we will describe the security of the cryptosystem.

4.1 Protocol

The information presented in this section is from both the original [15] and updated [14] papers by Hoffstein et al., as well as from an NTRU Public Key Cryptosystem tutorial released by Security Innovation, Inc [25].

4.1.1 Notations

The NTRU cryptosystem have three integer parameters (N, p, q) and four sets L_f, L_g, L_r, L_m of polynomials of degree $N-1$ with integer coefficients. These four sets will be defined in section 4.1.5. Integers p and q does not need to be prime, but it is assumed that $gcd(p, q) = 1$, and that q always is considerably larger than p. Operations are done in the ring $R = \mathbb{Z}[x]/(x^N - 1)$. An element $f \in R$ is written as a polyniomial or a vector,

$$f = \sum_{i=0}^{N-1} f_i x^i = [f_0, f_1, \ldots, f_{N-1}], \tag{4.1}$$

where $f_i \in \mathbb{Z}$. The symbol $\circledast$ is used to denote multiplication in R. This star multiplication is given explicitly as a cyclic convolution product,

$$f \circledast g = h \text{ with } h_k = \sum_{i=0}^{k} f_i g_{k-i} + \sum_{i=k+1}^{N-1} f_i g_{N+k-i} = \sum_{i+j \equiv k \pmod N} f_i g_j. \tag{4.2}$$

When doing a multiplication modulo q, reduce the coefficients moduluo q.

4.1.2 Key Creation

To use the NTRU cryptosystem, you need to create keys. So if Alice wants to create her key-pair, she has to randomly choose 2 polynomials $f \in L_f$ and $g \in L_g$. The polynomial f must satisfy the additional requirement that it have inverses modulo q and modulo p. The inverses are denoted by f_q^{-1} and f_p^{-1}:

$$f_q^{-1} \circledast f \equiv 1 \pmod q \tag{4.3}$$

and

$$f_p^{-1} \circledast f \equiv 1 \pmod p. \tag{4.4}$$

Next, Alice computes the polynomial

$$h = pf_q^{-1} \circledast g \pmod q. \tag{4.5}$$

The polynomial h is Alice's public key. The private key is the polynomial f, but f_p^{-1} should also be stored because it is used in decryption as well.

4.1.3 Encryption

Now that Alice got some keys, Bob wants to send her an encrypted message. He puts his message in the form of a polynomial m. The coefficients are chosen modulo p, for instance between $-p/2$ and $p/2$. It is common to have $p = 3$, so in this case the coefficients would be chosen from the set $\{-1, 0, 1\}$. Then he randomly chooses a polynomial $r \in L_r$ and uses Alice's public key h to compute

$$e = r \circledast h + m \pmod q. \tag{4.6}$$

The polynomial e is the encrypted message Bob sends to Alice.

4.1.4 Decryption

Alice receives the encrypted message e from Bob. She wants to decrypt it using her private key f. This is done efficiently if she stored the polynomial f_p^{-1} from the key generation. If not, she have to compute it again. To decrypt e, Alice computes

$$a = f \circledast e \pmod q, \tag{4.7}$$

where the coefficients of a is chosen in the interval from $-q/2$ to $q/2$. Alice then computes the polynomial

$$b = a \pmod p. \tag{4.8}$$

This is just reducing each of the coefficients of a modulo p. To finally recover the message, Alice uses the inverse polynomial f_p^{-1} to compute

$$c = f_p^{-1} \circledast b \pmod p. \tag{4.9}$$

The resulting polynomial c will be the original message m.

The decryption works because, although Alice doesn't know r and m, she actually performs the following computation:

$$\begin{aligned} a &= f \circledast e \pmod q \\ &= f \circledast (r \circledast h + m) \pmod q \\ &= f \circledast (r \circledast p f_q^{-1} \circledast g + m) \pmod q \\ &= pr \circledast g + f \circledast m \pmod q \end{aligned} \tag{4.10}$$

Since the polynomials r, g, f, m all have quite small coefficients, the coefficients of the products $r \circledast g$ and $f \circledast m$ are also quite small. We also have that the coefficients of the polynomial $pr \circledast g + f \circledast m$ already lie between $-q/2$ and $q/2$ because the prime p is small compared to q. This means that reducing the coefficients modulo q has no effect. So when Alice computes a by first multiplying $f \circledast e$ and then reducing the coefficients modulo q, a ends up being exactly equal to $pr \circledast g + f \circledast m$. Reducing a modulo p is therefore equal to reducing $pr \circledast g + f \circledast m$, and we get that

$$b = f \circledast m \pmod p. \tag{4.11}$$

To recover m, Alice now multiplies b with f_p^{-1} (because $f_p^{-1} \circledast f = 1 \pmod p$):

$$c = f_p^{-1} \circledast b = f_p^{-1} \circledast f \circledast m = m \pmod p. \tag{4.12}$$

Alice has now recovered the message m.

As mentioned earlier, the decryption relies on some probability. If the parameters are chosen correctly, there is an extremely high probability that the decryption procedure will recover the original message. It might be smart to include a few check bits in each message block to detect decryption errors, as they can happen. If an error happens when decrypting, it is often that the message is improperly centered. This can be fixed by choosing the coefficients of $a = f \circledast e \pmod q$ in a slightly different interval, for example from $-q/2+x$ to $q/2 + x$ for some small (positive or negative) value of x. If no value of x works, there is a gap failure. This makes it much harder to decrypt the message, but if the parameters is chosen correctly, this occurs so rarely that it can be ignored in practice [14].

4.1.5 Parameters

For the encryption and decryption to work properly, the parameters of the NTRU cryptosystem must be chosen correctly. This section presents how the parameter sets are constructed.

Notation and norm estimate

The width of an element $f \in R$ is defined to be

$$|f|_\infty = \max_{0 \le i \le N-1} \{f_i\} - \min_{0 \le i \le N-1} \{f_i\}. \tag{4.13}$$

This is a sort of an L^∞ norm on R. A centered L^2 norm on R is defined similarily:

$$|f|_2 = \left(\sum_{i=0}^{N-1} (f_i - \bar{f})^2 \right)^{\frac{1}{2}}, \quad \text{where } \bar{f} = \frac{1}{N} \sum_{i=0}^{N-1} f_i. \tag{4.14}$$

Proposition 4.1.1. *For any $\epsilon > 0$ there are constants $\gamma_1, \gamma_2 > 0$, depending on ϵ and N, such that for randomly chosen polynomials $f, g \in R$, the probability is greater than $1 - \epsilon$ that they satisfy*

$$\gamma_1 |f|_2 |g|_2 \le |f \circledast g|_\infty \le \gamma_2 |f|_2 |g|_2. \tag{4.15}$$

This proposition is useless from a practial viewpoint if the ratios γ_2/γ_1 are very large for small ϵ's. But for moderately large values of N and very small values of ϵ, the constants γ_1, γ_2 is not that extreme. This has been verified experimentally for a large number of parameter values [14].

Sample spaces

The space of messages L_m consist of all polynomials modulo p. Assuming p is odd, it is most convenient to construct it as

$$L_m = \left\{ m \in R \,:\, m \text{ has coefficients lying between } -\frac{1}{2}(p-1) \text{ and } \frac{1}{2}(p-1) \right\}. \tag{4.16}$$

The other sample spaces are described on the form

$$L(d_1, d_2) = \left\{ f \in R \,:\, f \text{ has } d_1 \text{ coefficients equal } 1,\, d_2 \text{ coefficients equal } -1, \text{ the rest } 0 \right\}. \tag{4.17}$$

Choose three positive integers d_f, d_g, d_r and set

$$\begin{aligned} L_f &= L(d_f, d_f - 1), \\ L_g &= L(d_g, d_g), \text{ and} \\ L_r &= L(d_r, d_r). \end{aligned} \tag{4.18}$$

Notice that $f \in L_f$, $g \in L_g$, and $r \in L_r$ have L^2 norms

$$\begin{aligned} |f|_2 &= \sqrt{2d_f - 1 - N^{-1}}, \\ |g|_2 &= \sqrt{2d_g}, \\ |r|_2 &= \sqrt{2d_r}. \end{aligned} \tag{4.19}$$

Decryption Criterion

For the decryption process to work, it is necessary that

$$|f \circledast m + pr \circledast g|_\infty < q. \tag{4.20}$$

This is virtually always true if the parameters are chosen so that

$$\begin{aligned} &|f \circledast m|_\infty \leq q/4 \text{ and} \\ &|pr \circledast g|_\infty \leq q/4, \end{aligned} \tag{4.21}$$

and in view of proposition 4.1.1, this suggests

$$\begin{aligned} &|f|_2 |m|_2 \approx q/4_{\gamma_2} \text{ and} \\ &|r|_2 |g|_2 \approx q/4p\gamma_2 \end{aligned} \tag{4.22}$$

for a γ_2 corresponding to a small value of ϵ.

4.1.6 Example

It often helps looking at a small example when trying to understand a cryptosystem, so this section will go through key creation, encryption, and decryption with some parameters that are too small to be used in practice, but still show how NTRU works. The example is from an NTRU Public Key Cryptosystem tutorial released by Security Innovation, Inc [25].

The parameters are:

- $N = 11$
- $q = 32$
- $p = 3$
- $L_f = (4, 3)$
- $L_g = (3, 3)$
- $L_r = (3, 3)$
- L_m consist of all polynomials with coefficents lying between -1 and 1.

Key Generation

Alice needs to choose the polynomials $f \in L_f$ and $g \in L_g$. Polynomial f is of degree 10 with four 1's and three -1's. Polynomial g is of degree 10 with four 1's and four -1's. Alice chooses:

$$f = -1 + x + x^2 - x^4 + x^6 + x^9 - x^{10}$$

$$g = -1 + x^2 + x^3 + x^5 - x^8 - x^{10}$$

Computing the inverses yields:

$$f_p^{-1} = 1 + 2x + 2x^3 + 2x^4 + x^5 + 2x^7 + x^8 + 2x^9 \pmod 3$$

$$f_q^{-1} = 5+9x+6x^2+16x^3+4x^4+15x^5+16x^6+22x^7+20x^8+18x^9+30x^{10} \pmod{32}$$

Creating the public key h:

$$\begin{aligned} h &= pf_q^{-1} \circledast g \\ &= 8 + 25x + 22x^2 + 20x^3 + 12x^4 + 24x^5 + 15x^6 + 19x^7 + 12x^8 + 19x^9 + 16x^{10} \pmod{32} \end{aligned}$$

Alice now have her private key (f, f_p^{-1}) and public key h.

Encryption

Bob wants to send Alice an encrypted message. The message is:

$$m = -1 + x^3 - x^4 - x^8 + x^9 + x^{10}$$

He must then choose the random polynomial $r \in L_r$. r must be of degree 10 with three 1's and three -1's:

$$r = -1 + x^2 + x^3 + x^4 - x^5 - x^7$$

The encrypted message e is then

$$\begin{aligned} e &= r \circledast h + m \\ &= 14 + 11x + 26x^2 + 24x^3 + 14x^4 + 16x^5 + 30x^6 + 7x^7 + 25x^8 + 6x^9 + 19x^{10} \pmod{32}. \end{aligned}$$

Decryption

Having received the encrypted message e, Alice computes

$$a = f \circledast e = 3 - 7x - 10x^2 - 11x^3 + 10x^4 + 7x^5 + 6x^6 + 7x^7 + 5x^8 - 3x^9 - 7x^{10} \pmod{32}.$$

The coefficients of a are chosen between -15 and 16, not 0 and 31. Reducing the coefficients of a modulo 3 yields

$$b = a = -x - x^2 + x^3 + x^4 + x^5 + x^7 - x^8 - x^{10} \pmod{3}.$$

The last step is to multiply b with f_p^{-1}:

$$c = f_p^{-1} \circledast b = -1 + x^3 - x^4 - x^8 + x^9 + x^{10} \pmod{3}$$

We can see that $c = m$, so Alice has successfully recovered the original message.

4.2 Security

As already mentioned, the security of NTRU relies on the problem of finding short vectors in a lattice. It is therefore susceptible to attacks from algorithms solving that problem. In this section, a few attacks against NTRU will be described.

4.2.1 Security levels

In the original paper where the NTRU cryptosystem was first presented, the authors presented three sets of parameters which was updated in [14].

Moderate Security

This level of security was intented for situations in which the intrinsic value of any individual message is small and keys are changed with reasonable frequency. The parameters are set to the following:

$$(N, p, q) = (167, 3, 128)$$

$$L_f = L(61, 60)$$
$$L_g = L(20, 20)$$
$$L_r = L(18, 18)$$

This means that f is chosen with 61 1's and 60 -1's (i.e., $d_f = 61$), g is chosen with 20 1's and 20 -1's (i.e., $d_g = 20$), and r is chosen with 18 1's and 18 -1's (i.e., $d_r = 18$) The security level of these parameters are based on the meet-in-the-middle attack (which will be described later in this section) and are as follows:

$$\text{Key security} = 2^{82.9}$$
$$\text{Message security} = 2^{77.5}$$

High Security

$$(N, p, q) = (263, 3, 128)$$
$$L_f = L(50, 49)$$
$$L_g = L(24, 24)$$
$$L_r = L(16, 16)$$
$$\text{Key security} = 2^{110.6}$$
$$\text{Message security} = 2^{82.1}$$

Highest Security

$$(N, p, q) = (503, 2, 256)$$
$$L_f = L(216, 215)$$
$$L_g = L(72, 72)$$
$$L_r = L(55, 55)$$
$$\text{Key security} = 2^{285}$$
$$\text{Message security} = 2^{170}$$

Since these parameters was presented in 1999, they are not really up to date. On the NTRU GitHub website [1], they operate with four levels of security: 112, 128, 192, and 256 bits.

4.2.2 Brute force attacks

The simplest attack against cryptographic systems is to do a search over all possible keys or messages. For NTRU, there's a few different ways to do this. An attacker can try all possible $f \in L_f$ and test if $f \circledast h \pmod{q}$ has small entries, or try all $g \in L_g$ and test if $g \circledast h^{-1} \pmod{q}$ has small entries. By doing this, the attacker can recover the private key. If the attacker wants to recover a message directly, he can try all $r \in L_r$ and test if $e - r \circledast h \pmod{q}$ has small entries. Since L_g will be smaller than L_f, the key security is determined by the number of elements in L_g. The message security is determined by the number of elements in L_r [14].

4.2.3 Meet-in-the-middle attacks

There exists meet-in-the-middle attacks against both r and f. An attacker would split f in half, for instance $f = f_1 + f_2$, and then match $f_1 \circledast e$ against $-f_2 \circledast e$, looking for (f_1, f_2) so that the corresponding coefficients have approximately the same value [14]. This attack cuts the search time of the brute force attack by the square root, meaning that for a security level of 256 bits, L_f, L_g, L_r must contain around 2^{512} elements.

4.2.4 Lattice attacks

Finding the shortest vector, also called lattice reduction, is one way to attack NTRU. The shortest vector could be searched for in a brute force attack, but with large enough dimensions on the lattice, this is not possible in practice. The LLL-algorithm, as mentioned in chapter 3, is a polynomial time algorithm for reducing lattices and finding short vectors, but this algorithm will also take too long time finding the shortest vector provided that the shortest vector is not too much smaller than the expected length of the smallest vector [14]. Hoffstein et al. presented some estimated breaking times in their paper. For running an improved version of the LLL-algorithm on a 400 MHz Celeron machine, it was estimated that it would take $1.638 \cdot 10^{11}$ seconds to break NTRU 167, $3.634 \cdot 10^{19}$ seconds to break NTRU 263, and $2.663 \cdot 10^{40}$ seconds to break NTRU 503 [14]. That is almost 5200 years for the lowest level of security.

4.3 Performance

The security levels also change the size of the keys and data.

Moderate Security

$$\text{Private key} = 530 \text{ bits}$$
$$\text{Public key} = 1169 \text{ bits}$$
$$\text{Plaintext} = 187 \text{ bits}$$

High Security

$$\text{Private key} = 834 \text{ bits}$$
$$\text{Public key} = 1841 \text{ bits}$$
$$\text{Plaintext} = 335 \text{ bits}$$

Highest Security

$$\text{Private key} = 1595 \text{ bits}$$

$$\text{Public key} = 4024 \text{ bits}$$

$$\text{Plaintext} = 628 \text{ bits}$$

For the four security levels presented on the NTRU GitHub webpage (112, 128, 192 and 256 bits), the key sizes are 5951, 6743, 9757, and 12881 bits, respectively. This is a quite large increase from the original values, but is to be expected with increased computing power and more optimised attacks.

When it comes to how fast NTRU is, it has been reported to be considerable faster than other public key cryptosystems. For NTRU 167 ($n = 167$) with public key size of 1169 bits, the key creation runs in 4 milliseconds, encryption runs in 5941 blocks/second, and decryption runs in 2818 blocks/second. A block is a single message block. The algorithm is running on a 300 MHz Pentium II operating under Linux. For NTRU 263 with public key size 1841 bits, we have key creation in 7.5 milliseconds, encryption in 3676 blocks/second, and decryption in 1619 blocks/second. For NTRU 503 with public key size 4024, we have key creation in 17.3 milliseconds, encryption in 1471 blocks/second, and decryption in 608 blocks/second [14].

Same as with the key sizes, these numbers are outdated. More updated statistics are presented on the GitHub site, for the same security levels as the key sizes. These statistics use operations per second, so it is not split up in key creation, encryption, and decryption as the ones in [14]. For a security level of 112 bits, with key size of 5951 bits, it manages to do 2284 operations/second with the standard implementation. For 128 bits with key size 6743 bits, it does 1896 operations/second, for 192 bits with key size 9759 bits, it does 1034 operations/second, and for 256 bits with key size 12881 bits, it does 638 operations/second [1]. It is unclear how one operation compares to encrypting or decrypting one single message block, so it is hard to compare

	NTRU 167	NTRU 263
Public key size (bits)	1169	1841
Key creation (milliseconds)	4	7.5
Encryption (blocks/second)	5941	3676
Decryption (blocks/second)	2818	1619

Table 4.1: Performance of NTRU running on a 300 MHz Pentium II.

	112 bits	128 bits	192 bits	256 bits
Key size (bits)	5951	6743	9759	12881
Operations/second	2284	1896	1034	638

Table 4.2: Performance of NTRU presented on [1].

	Ordinary ternary polynomials	Product-form ternary polynomials
Encryption (operations/second)	25025	221845
Decryption (operations/second)	24331	

Table 4.3: Performance of NTRU running on an Nvidia GTX280 GPU.

the numbers. It is also not mentioned what hardware the algorithm runs on, but it is safe to assume it is running on hardware much better than the 300 MHz Pentium II.

Because of how NTRU is constructed, it is possible to parallelize the algorithm. Jens Hermans, Frederik Vercauteren, and Bart Preneel released a paper presenting results from running a parallelized version of NTRU on a Graphical Processing Unit (GPU) [13]. General Purpose GPUs are well suited for running parallelized algorithms because they contain a large number of processor cores. Hermans et al. used an Nvidia GTX280 GPU with 240 cores running at 650 MHz, and ran the algorithm with a security level of 256 bits. Using ordinary ternary polynomials, they managed to do 25025 operations/second for encryption and 24331 operations/second for decryption. When using product-form ternary polynomials, which minimize the need to access memory, the implementation did 221845 operations/second for encryption [13]. You will only get this performance when doing a large number of operations. With ordinary ternary polynomials, the algorithm did 20000 operations, while with product-form ternary polynomials the algorithm did 2^{16} operations. Doing one operation is not much faster than running a non-parallelized implementation on a CPU.

4.4 Summary

It is now over 20 years since the first version of NTRU was presented, and it is one of the more well-known lattice-based cryptosystems. After the release of the open-source reference implementation, it has seen some use in open-source applications [31]. One aspect of NTRU that is a bit negative is that the mathematics of NTRU is more advanced than Diffie-Hellman and RSA, so it is harder to understand, especially for people with no mathematical background. This can increase the risk for bad implementations of the cryptosystem. Therefore it is important to use the standarized version of NTRU. NTRUEncrypt and NTRUSign have been included in the IEEE Standard 1363.1 from 2008.

NTRU looks like a possible replacement for the standard public key cryptosystems. It has been analyzed and tested in over 20 years. The development and public adoption of the system have probably been hindered a bit by the authors patenting it, but with the open-source reference implementation, this is no longer a problem for open-source applications. NTRU is more advanced than Diffie-Hellman and RSA, but the encryption and decryption operations are not too hard to understand. As we will see in the next chapter, there are lattice-based cryptosystems that are even more advanced mathematically than NTRU.

Chapter 5

Ring-LWE

One of the more common problems to base a lattice-based cryptographic system on, is the *learning with errors over rings* (ring-LWE) problem. In 2014, Chris Peikert released a paper giving efficient and practical lattice-based protocols for key transport, encryption, and authenticated key exchange [22]. These protocols are based on the ring-LWE problem, and their security is provable.

The paper presents a new reconciliation mechanism for transforming approximate agreement to exact agreement. This reconciliation mechanism is used to create a new passively secure *key encapsulation mechanism* (KEM), which is also transformed into an active KEM. An authenticated key exchange (AKE) protocol is also described.

In this chapter, we will first describe the necessary cryptographic and mathematical background and the ring-LWE problem. Then the passively secure KEM, active KEM, and the AKE will be described. The information in this chapter is from Peikert's paper on lattice cryptography for the internet [22].

5.1 Background and Ring-LWE

The ring-LWE problem has a lot of advanced mathematical background information. The most important bits will be described in this section, but a full description of the mathematical background of the ring-LWE problem can be found in Peikert's paper on ring-LWE [22].

Cyclotomic Rings

Similar to NTRU and as the name implies, we are working with polynomial rings. The rings used in ring-LWE are cyclotomic rings. Cyclotomic rings are obtained from cyclotomic fields, which are number fields obtained by adjoining a complex primitive root of unity to $\mathbb{Q}$, the field of rational numbers. A number field is a finite extension of $\mathbb{Q}$, and a root of unity is any complex number that gives 1 when raised to some positive integer power m. The mth cyclotomic field $\mathbb{Q}(\zeta_m)$ (where $m > 2$) is obtained by adjoining a primitive mth root of unity ζ_m to the rational numbers. A number ring is the algebraic integers in a number field. An algebraic integer is a complex number that is a root of some polynomial whose leading coefficient is 1 (monic polynomial) with coefficients in $\mathbb{Z}$. So a cyclotomic ring is the ring of algebraic integers from the corresponding cyclotomic field.

For Peikert's ring-LWE we then have that, for a positive integer index m, $K = \mathbb{Q}(\zeta_m)$ denotes the mth cyclotomic field and $R = \mathbb{Z}[\zeta_m] \subset K$ denotes the mth cyclotomic ring. For any integer modulus $q \geq 1$, let R_q denote the quotient ring R/qR. More details on cyclotomic rings and their properties in ring-LWE is found in Peikert's paper [22].

Error Distributions

To hide information in ring-LWE, small random values are added when encrypting. These values are secret, which we will see in the protocols explained in section 5.2 and 5.3. The secret values are the errors which are referred to in the name *learning with errors.* You are trying to learn information from data that have errors, in this case small secret values that have been added to the original value.

The error values are taken from error distributions, which are Gaussian like distributions over the number field K. For $r > 0$, the Gaussian distribution D_r over $\mathbb{R}$ with parameter r has probability distribution function $\exp(-\pi x^2/r^2)/r$. This means that all possible error values are distributed according to the distribution function. The distribution we are retrieving the error values from is denoted with χ, and $e \to \chi$ means choosing the error value e from the distribution χ.

Ring-LWE

Here the ring-LWE probability distribution and computational problem is described. The problem is presented in its discretized, "normal" form, where all quantities are from R or $R_q = R/qR$, and the secret is drawn from the error distribution.

Definition 5.1.1. Ring-LWE Distribution *For an $s \in R$ and a distribution χ over R, a sample from the ring-LWE distribution $A_{s,\chi}$ over $R_q \times R_q$ is generated by choosing $a \leftarrow R_q$ uniformly at random, choosing $e \leftarrow \chi$, and outputting $(a, b = a \cdot s + e)$.*

This definition (5.1.1) defines the ring-LWE distribution and tells us how to choose the variables needed for using ring-LWE. We will see more of this when the actual protocol is described in later sections.

Definition 5.1.2. Ring-LWE Decision *The decision version of the ring-LWE problem, denoted R-DLWE$_{q,\chi}$, is to distinguish with non-negligible advantage between independent samples from $A_{s,\chi}$ where $s \leftarrow \chi$ is chosen once and for all, and the same number of uniformly random and independent samples from $R_q \times R_q$.*

This definition (5.1.2) defines the ring-LWE decision problem. This is the problem that ring-LWE based cryptosystems base theiir security on. The decision problem is to distinguish between samples (as chosen in definition 5.1.1) where $s \rightarrow \chi$ is chosen and samples that are random. In other words, being able to choose between constructed samples and random samples.

Theorem 5.1.1. *Let R be the mth cyclotomic ring, having dimension $n = \varphi(m)$. Let $\alpha = \alpha(n) < \sqrt{\log n/n}$, and let $q = q(n), q = 1 \mod m$ be a $\text{poly}(n)$-bounded prime such that $\alpha q \geq \omega(\sqrt{\log n})$. There is a $\text{poly}(n)$-time quantum reduction from $\tilde{O}(\sqrt{n}/\alpha)$-approximate SIVP (or SVP) in ideal lattices in R to solving R-DLWE$_{q,\chi}$ given only $\ell - 1$ samples, where $\chi - \lfloor \psi \rceil$ and ψ is the Gaussian distribution $(\hat{m}/g) \cdot D_{\xi q}$ for $\xi = \alpha \cdot (n\ell/\log n\ell)^{1/4}$.*

This theorem (5.1.1) connects the ring-LWE decision problem to lattices. It states that there is a polynomial time quantum reduction from the shortest independent vectors problem (or shortest vector problem, described in chapter 3) to the ring-LWE decision problem. This means that if you are able to solve the ring-LWE decision problem, you are also able to solve SIVP and

SVP. Solving SIVP and SVP is believed to be hard, so solving the ring-LWE decision problem must be at least as hard, making it ideal for a cryptosystem. The theorem was presented in [17] by Vadim Lyubashevsky, Oded Regev, and Peikert. They also present a proof for the theorem, but it is too technical to present it here.

Reconciliation Mechanism

The way ring-LWE is constructed, the shared key between two parties are only approximately equal. A mechanism is then needed for the two parties to extract the exact shared key. This mechanism is referred to as the reconciliation mechanism (reconciliation means the action of making one view or belief compatible with another). Peikert presents a reconciliation mechanism used for a bandwidth-efficient method for two parties to agree on a secret bit. The method directly produces an unbiased key. The reconciliation mechanism works by creating intervals between 0 and q, and transforms all the coefficients of the polynomial into 0 or 1. Half of the intervals yields 0, the other half yield 1. So if a coefficient is in an interval yielding 0, the coefficient is set to 0, and vice versa. Since all operations are done modulo q, all coefficients are between 0 and q. All coefficients will therefore be set to either 0 or 1. The reconciliation mechanism is unbiased as long as q is even, because you can create intervals of the same size. This is not possible with an odd q, and the mechanism will output a value that is biased and unsuitable for as key material. Peikert avoids this by temporarily scaling up to $2q$ in the case of odd q. A small amount of extra randomness is also added, which ensures that the reconciliation mechanism produces an unbiased key. The reconciliation mechanism is denoted by rec and the upscaling used in the odd case is denoted dbl.

5.2 From passive KEM to active KEM

The main focus of Peikert's paper is to present an authenticated key exchange (AKE) protocol based on ring-LWE. This is done by first constructing a passive key encapsulation mechanism (KEM), which is secure against passive attacks. The passive KEM is then transformed to an active KEM using the Fujisaki-Okamoto transformation, which is then used to construct the AKE.

5.2.1 Passively Secure KEM

This system is constructed explicitly as a KEM, meaning that the encapsulated key is not chosen by either party, but rather the sender and receiver "approximately agree" on a pseudorandom value in R_q using ring-LWE and use the reconciliation mechanism to derive the ephemeral key from it.

Construction

The KEM is parameterized by:

- A positive integer m specifying the mth cyclotomic ring R of degree $n = \varphi(m)$.
- A positive integer modulus q which is coprime with every odd prime dividing m, so that $g \in R$ is coprime with q. For efficiency and provable security, typically take q to be prime and 1 modulo m.
- A discretized error distribution χ.

The algorithms for the passive KEM are:

- KEM1.Setup(): choose $a \leftarrow R_q$ and output $pp = a$.
- KEM1.Gen($pp = a$): choose $s_0, s_1 \leftarrow \chi$, let $b = a \cdot s_1 + s_0 \in R_q$, and output public key $pk = b$ and secret key $sk = s_1$.
- KEM1.Encaps($pp = a, pk = b$): choose independent $e_0, e_1, e_2 \leftarrow \chi$. Let $u = e_0 \cdot a + e_1 \in R_q$ and $v = g \cdot e_0 \cdot b + e_2 \in R_q$. Let $\bar{v} \leftarrow \mathrm{dbl}(v)$ and output the encapsulation $c = (u, v' = \langle \bar{v} \rangle_2) \in R_q \times R_2$ and key $\mu = \lfloor \bar{v} \rceil_2 \in R_2$.
- KEM1.Decaps($sk = s_1, c = (u, v')$): compute $w = g \cdot u \cdot s_1 \in R_q$ and output $\mu = \mathrm{rec}(w, v') \in R_2$.

The KEM is passively secure (IND-CPA secure), meaning it is secure against chosen-plaintext attacks. Peikert presents a proof of this, which can be seen in [22].

5.2.2 Actively Secure KEM

When creating a cryptosystem you want it to not only be passively secure, but also secure against adaptive chosen-ciphertext attacks, also called actively secure. Peikert uses the Fujisaki-Okamoto transformation to transform the passively secure KEM to an actively secure encryption and KEM scheme, which can be used as an alternative to, e.g., RSA-based actively secure key encapsulation. The Fujisaki-Okamoto transformation [8] is a method for transforming cryptosystems with "weak" security to cryptosystems with much stronger security, i. e. from IND-CPA to IND-CCA.

Construction

The actively secure encryption scheme PKC2 is parameterized by:

- An integer N, the bit length of the messages that PKC2 will encrypt.
- An asymmetric encryption scheme PKC with message space $\{0,1\}^n$, where PKC.Enc uses at most L uniformly random bits (i.e., PKC.Enc$(pp, pk, \cdot; r)$ is a deterministic function on $\{0,1\}^n$ for any fixed pp, pk, and coins $r \in \{0,1\}^L$), e.g., the encryption scheme induced by KEM1.
- A cryptographic pseudorandom generator PRG : $\{0,1\}^\ell \to \{0,1\}^L$, for some seed length ℓ.
- Hash functions $G : \{0,1\}^n \to \{0,1\}^N$ and $H : \{0,1\}^{n+N} \to \{0,1\}^\ell$, modelled as independent random oracles.

PKC2 is defined as follows:

- PKC2.Setup(): let $pp \leftarrow \text{PKC.Setup}()$ and output pp.
- PKC2.Gen(pp): let $(pk, sk) \leftarrow \text{PKC.Gen}(pp)$ and output public key pk and secret key sk.
- PKC2.Enc(pp, pk, μ): choose $\sigma \leftarrow \{0,1\}^n$, let $c = \text{PKC.Enc}(pp, pk, \sigma; \text{PRG}(H(\sigma\|\mu)))$ and $w = G(\sigma) \oplus \mu$, and output the ciphertext $c\|w$.
- PKC2.Dec$(sk, (c, w))$: compute $\sigma = \text{PKC.Dec}(sk, c)$ and $\mu = G(\sigma) \oplus w$, and check whether $c \overset{?}{=} \text{PKC.Enc}(pp, pk, \sigma; \text{PRG}(H(\sigma\|\mu)))$. If so, output μ, otherwise output $\perp$.

PKC2 is actively secure (IND-CCA secure) if PRG is a secure pseudorandom generator and G and H are modeled as random oracles. This means that it is secure against chosen-ciphertext attacks.

5.3 Authenticated Key Exchange

An authenticated key exchange (AKE) protocol is a bit different from a key exchange protocol. With a key exchange protocol (e.g., Diffie-Hellman) two parties can agree on a shared secret. But this is only secure against a passive adversary who only reads the network traffic. An AKE, on the other hand, authenticates the parties' identites to each other and provides a consistent view of the completed protocol to the peers, even in the presence of an active adversary. Peikert presents an AKE where an abstract IND-CPA-secure KEM (which can be instantiated by the lattice-based KEM1) is used for key agreement. The protocol is called Σ_0' and is a slight modification of the Σ_0 protocol presented by Canetti and Krawczyk [5].

5.3.1 Protocol

Σ_0' is parameterized by a digital signature scheme SIG, a key-encapsulation mechanism KEM with key space K, a pseudorandom function $F : K \times \{0,1\} \leftarrow K'$, and a message authentication code MAC with key space K' and message space $\{0,1\}^*$. A successful execution of the protocol outputs a secret key in K'. It is assumed that each party has a long-term signing key for SIG and that trusted public parameters pp for KEM have been generated by a trusted party using KEM.Setup, and are available for all parties.

1. Start message ($I \to R$):

   (sid, pk_I)

   The protocol is activated by the initiator ID_I with a session identifier sid, which must be distinct from all those of prior sessions initiated by ID_I. The initiator generates a new key pair $(pk_I, sk_I) \leftarrow \mathrm{KEM.\,Gen}(pp)$, stores it as the state of the session $(\mathrm{ID}_I, \mathrm{sid})$, and sends the above message to the responder.

2. Response message ($R \to I$):

   $(\mathrm{sid}, c, \mathrm{ID}_R, \mathrm{SIG.\,Sign}_R(1, \mathrm{sid}, pk_I, c)\mathrm{MAC.\,Tag}_{k_1}(1, \mathrm{sid}, \mathrm{ID}_R))$

When a party ID_R receives a start message (sid, pk_I), if the session identifier sid was never used before at ID_R, the party activates session sid as responder. It generates an encapsulation and key $(c, k) \leftarrow \mathrm{KEM.Encaps}(pp, pk_I)$, derives $k_0 = F_k(0)$ and $k_1 = F_k(1)$, and erases the values pk_I and k from its memory, saving (k_0, k_1) as the state of the session. It generates and sends the above response message, where $\mathrm{SIG.Sign}_R$ is computed using its long-term signing key, and MAC.Tag is computed using key k_1.

3. Finish message $(I \rightarrow R)$:

   $(\mathrm{sid}, \mathrm{ID}_I, \mathrm{SIG.Sign}_I(0, \mathrm{sid}, c, pk_I), \mathrm{MAC.Tag}_{k_1}(0, \mathrm{sid}, \mathrm{ID}_I))$

   When party ID_I receives the response message $(\mathrm{sid}, c, \mathrm{ID}_R, \sigma_R, \tau_R)$ having session identifier sid, it looks up the state (pk_I, sk_I) associated with session sid and computes $k = \mathrm{KEM.Decaps}(sk_I, c)$ and $k_0 = F_k(0), k_1 = F_k(1)$. It then retrieves the signature verification key of ID_R and uses that key to verify the signature σ_R on the message tuple $(1, \mathrm{sid}, pk_I, c)$, and also verifies the MAC tag τ_R on the message tuple $(1, \mathrm{sid}, \mathrm{ID}_R)$ under key k_1. If either verification fails, the session is aborted, its state is erased, and the session output is $(\mathrm{abort}, \mathrm{ID}_I, \mathrm{sid})$. If both verifications succeed, then ID_I completes the session as follows: it generates and sends the above finish message where $\mathrm{SIG.Sign}_I$ is computed using its long-term signing key, and MAC.Tag is computed using key k_1. It then produces public session output $(\mathrm{ID}_I, \mathrm{sid}, \mathrm{ID}_R)$ and session secret output k_0, and erases the session state.

4. Responder completion:

   When party ID_R receives the finish message $(\mathrm{sid}, \mathrm{ID}_I, \sigma_I, \tau_I)$ having session identifier sid, it looks up the sate (k_0, k_1) associated with session sid. It then retrieves the signature verification key of ID_I and uses that key to verify the signature σ_I on the message tuple $(0, \mathrm{sid}, c, pk_I)$, and also verifies the MAC tag τ_I on the message tuple $(0, \mathrm{sid}, \mathrm{ID}_I)$ under key k_1. If either verification fails, the session is aborted, its state is erased, and the session output is $(\mathrm{abort}, \mathrm{ID}_R, \mathrm{sid})$. If both verifications succeed, then ID_R completes the session with public session output $(\mathrm{ID}_R, \mathrm{sid}, \mathrm{ID}_I)$ and secret session output k_0, and erases the session state.

5.3.2 Security

Theorem 5.3.1. *The Σ_0' protocol is SK-secure in the post-specified peer model of [5], assuming that SIG and MAC are existentially unforgeable under chosen-message attack that KEM is IND-CPA secure, and that F is a secure pseudorandom function.*

The term "SK-secure" is used in [5] and will be described briefly here. The attacker is a polynomial-time machine with full control of the communication lines between parties. All the attacker's actions can be decided by the attacker in a fully adaptive way. The "success" of the attacker is measured via its ability to distinguish real session keys from random values. Such an attacker is called an *SK-attacker.* A key-exchange protocol π is called *SK-secure* if for all SK-attackers A running against π it holds:

1. If two uncorrupted parties complete matching sessions in a run of protocol π under attacker A then, except for a negligible probability, the session key output in these sessions is the same.

2. A succeeds with probability not more than $1/2$ plus a negligible fraction.

Since Σ_0' is a slight modification of the Σ_0 presented in [5], the theorem can be proved by slightly modifying the proof given in [5].

Two properties must be fullfilled:

1. Correctness: equality of the secret outputs when two uncorrupted parties $\mathrm{ID}_I, \mathrm{ID}_R$ complete matching sessions with respective public outputs $(\mathrm{ID}_I, \mathrm{sid}, \mathrm{ID}_R), (\mathrm{ID}_R, \mathrm{sid}, \mathrm{ID}_I)$.

2. Secrecy: no efficient attacker (in the post-specified peer model) can distinguish a real response to a test-session query from a uniformly random response, with non-negligible advantage.

For property 1 it suffices to show that both parties compute the same decapsulation key k. By the correctness of KEM and the security of the signature scheme, this is guaranteed.

Property 2 requires to modify the proof from [5]. Since Σ_0 uses Diffie-Hellman and the DDH assumption, it is enough to change the proof to use the KEM instead. In the proof, a distinguisher for the DDH problem is constructed, i.e., it get as input a tuple (g, g^x, g^y, g^z) where either $z = xy$

or z is uniformly random modulo the order of the group generated by g. In the modified proof, this is replaced by a distinguisher for the IND-CPA security of KEM, i.e., it gets as input a tuple (pp, pk, c, k) where either k is the decapsulation of ciphertext c, or is uniformly random in the key space K. With these changes, the proof remains valid.

5.4 Summary

This ring-LWE-based cryptosystem is very new compared to other types of cryptosystems. Peikert's paper was released in 2014, almost 20 years after the first version of the NTRU cryptosystem. Even so, ring-LWE provides strong, provable security against quantum computers. Peikert does not provide any implementation of the cryptosystem, but as we will see in the next chapters, there are several implementations based on Peikert's work.

It was mentioned in chapter 4 that NTRU is more advanced than Diffie-Hellman and RSA, making it harder to understand and possibly use. Ring-LWE, or more specific, Peikert's description of ring-LWE is even more complicated. This might make it harder for the cryptosystem to gain a widespread adoption. A move towards an easier description of the system would probably benefit the system in the long run.

Chapter 6

Post-quantum key exchange

Peikert did not present any implementation of a cryptosystem based on ring-LWE, only the theory behind one. But others have implemented such systems. In this chapter, two of them will be presented. First the Post-quantum key exchange for the TLS protocol from the ring learning with errors problem by Bos et al. will be discussed [4]. Then Post-quantum key exchange - a new hope by Alkim et al. will be discussed [3]. The latter system is based on the former.

6.1 Post-quantum Key Exchange for TLS

One implementation of a ring-LWE based cryptosystem was presented by Bos, Costello, Naehrig, and Stebila [4]. The system is aimed at replacing the traditional number-theoretic key exchange in the Transport Layer Security (TLS) protocol. It is implemented in the C programming language and targeting the 128-bit security level. The information presented in this section is from the paper by Bos et al. [4].

6.1.1 Changes from Peikert

As described in chapter 5, Peikert presents a passively secure KEM which is transformed to an actively secure KEM based on the Σ_0 protocol. The Post-quantum key exchange is based on Peikert's passively secure KEM, but is constructed more like a Diffie-Hellman cryptosystem. One can also describe it as a reformulation of Peikert's KEM. Phrasing the system as a

Diffie-Hellman-like protocol makes it easier to integrate into existing network protocols like TLS, because these protocols already are Diffie-Hellman-based. To create a Diffie-Hellman-like protocol, Bos et al. defines the decision R-LWE problem a bit different than Peikert:

Definition 6.1.1. *Let $R = \mathbb{Z}[X]/(\Phi_m(X))$, $n = 2^l, l > 0$, $m = 2n$, q an integer modulus, and $R_q = R/qR \cong \mathbb{Z}_q[X]/(X^n + 1)$ with $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$. Let χ be a distribution over R and let $s \overset{\$}{\leftarrow} \chi$. Define $O_{\chi,s}$ as the oracle which does the following:*

1. *Sample $a \overset{\$}{\leftarrow} U(R_q), e \overset{\$}{\leftarrow} \chi$,*
2. *Return $(a, as + e) \in R_q \times R_q$.*

The decision R-LWE problem *for n, q, χ is to distinguish $O_{\chi,s}$ from an oracle that returns uniform random samples from $R_q \times R_q$. In particular, if A is and algorithm, define the advantage*

$$\mathrm{Adv}^{\mathrm{drlwe}}_{n,q,\chi}(A) = |\Pr(s \overset{\$}{\leftarrow} \chi; A^{O_{\chi,s}}(\cdot) = 1) - \Pr(A^{U(R_q \times R_q)}(\cdot) = 1)|.$$

If χ is a probability distribution over R, where R is the ring of integers of the m-th cyclotomic number field, then $x \overset{\$}{\leftarrow} \chi$ denotes sampling $x \in R$ according to χ.

The *decision Diffie-Hellman-like* problem is defined as follows:

Definition 6.1.2. *Let q, n, χ be R-LWE parameters. The* decision Diffie-Hellman-like *(*ddhℓ*) problem for q, n, χ is to distinguish DH-like tuples with a real shared secret from those with a random value, given reconciliation information. If A is an algorithm, define*

$$\mathrm{Adv}^{\mathrm{ddh}\ell}_{q,n,\chi}(A) = |\Pr(A(a, b, b', c, k) = 1) - \Pr(A(a, b, b', c, k') = 1)|,$$

where $a \overset{\$}{\leftarrow} U(R_q), s, s', e, e', e'' \overset{\$}{\leftarrow} \chi, b \leftarrow as+e, b' \leftarrow as'+e', v \leftarrow bs'+e'', \bar{v} \overset{\$}{\leftarrow}$ $\mathrm{dbl}(v), c \leftarrow \langle \bar{v} \rangle_{2q,2}, k \leftarrow \lfloor \bar{v} \rceil_{2q,2}$, and $k' \overset{\$}{\leftarrow} U(\{0,1\}^n)$.

An overview of the unauthenticated Diffie-Hellman-like key exchange is shown in figure 6.1.1.

Public parameters		
Decision R-LWE parameters q, n, χ		
$a \xleftarrow{\$} U(R_q)$		
Alice		**Bob**
$s, e \xleftarrow{\$} \chi$		$s', e' \xleftarrow{\$} \chi$
$b \leftarrow as + e \in R_q$	$\xrightarrow{b}$	$b' \leftarrow as' + e' \in R_q$
		$e'' \xleftarrow{\$} \chi$
		$v \leftarrow bs' + e'' \in R_q$
		$\bar{v} \xleftarrow{\$} \mathrm{dbl}(v) \in R_{2q}$
	$\xleftarrow{b',c}$	$c \leftarrow \langle \bar{v} \rangle_{2q,2} \in \{0,1\}^n$
$k_A \leftarrow \mathrm{rec}(2b's, c) \in \{0,1\}^n$		$k_B \leftarrow \lfloor \bar{v} \rceil_{2q,2} \in \{0,1\}^n$

Figure 6.1: Unauthenticated Diffie-Hellman-like key exchange.

6.1.2 Performance

To aquire a security level of at least 128 bit, the parameters are set to the following:

- $n = 1024$
- $q = 2^{32} - 1$
- $\sigma = 8/\sqrt{2\pi} \approx 3.192$

Some performance testing has been done with the system. The testing involved two computers: a "client" and a "server". The client computer had an Intel Core i5 (4570R) processor with 4 cores running at 2.7 GHz each. The server computer had an Intel Core 2 duo (E6550) processor with 2 cores running at 2.33 GHz each. The software was compiled for the x86_64 architecture with -03 optimizations using `llvm` 5.1 (`clang` 503.0.40) on the client computer and `gcc` 4.7.2 on the server computer.

OpenSSL cryptographic primitive performance

OpenSSL cryptographic primitive performance where measured by running `openssl speed`. The following numbers are average runtime in milliseconds. For the client computer, key generation ran in 0.9 milliseconds and generating

	Client	Server
Key generation	0.9	1.7
Shared secret	0.5	0.4
Total	1.4	2.1

Table 6.1: OpenSSL cryptographic primitive performance in milliseconds.

	R-LWE ECDSA	R-LWE RSA
Connection time (standard deviation)	45.6 ms (0.90)	54.0 ms (1.49)
Size of handshake	9469 bytes	10476 bytes

Table 6.2: OpenSSL/Apache TLS performance.

shared secret ran in 0.5 milliseconds, making the total runtime for ring-LWE 1.4 milliseconds. For the server computer, key generation ran in 1.7 milliseconds and generating shared secret ran in 0.4 milliseconds, making the total runtime for ring-LWE 2.1 milliseconds.

OpenSSL/Apache TLS performance

Performance within the context of HTTP connections over TLS was also measured . The server was running Apache `httpd` 2.4.10 with the prefork module for multi-threading. The client and server computers were connected over an isolated local area network with less than 1 millisecond ping time. The `http_load` tool was used to create many HTTP connections in parallel using OpenSSL for TLS.

The connection time (from when the client opens the TCP connection to the server's IP address to when the client starts to receive the first packet of application data) was measured to be 45.6 milliseconds mean value, with 0.90 as standard deviation, for ECDSA with ring-LWE. For RSA with ring-LWE it was 54.0 milliseconds with 1.49 as standard deviation. The size of the handshake was 9469 and 10476 bytes, respectively. The number of simultaneous connections were also measured. It was tested for four different payload sizes: 1 B, 1 KiB, 10 KiB, and 100 KiB. For ECDSA with ring-LWE, the results was 507.5 (1.7 standard deviation), 505.9 (2.1), 490.9 (0.9), and 397.6 (1.4), respectively. These are average median values of 5 runs of 100 seconds for each payload size.

6.2 A new hope

Another implementation of a ring-LWE cryptosystem is the *NewHope* system [3]. It was presented by Alkim, Ducas, Pöppelmann, and Schwabe, and builds on the scheme presented by Bos et al. The NewHope system has new parameters and better suited error distribution. Information in this section is from the paper of Alim et al.

6.2.1 Changes from Post-quantum Key Exchange for TLS

As mentioned, the NewHope system are based on the scheme presented by Bos et al., but Alkim et al. have done some changes and possibly some improvements.

Parameters

Most of the parameters are kept the same, but some changes are made. The dimension $n = 1024$ is the same to be able to achieve appropriate long-term security. Since polynomial arithmetic is fast and scale well, the choice of n is acceptable from a performance point of view. Polynomials is still defined in the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$. The modulus is changed to $q = 12289$. It is the smallest prime for which it holds that $q \equiv 1 \pmod{2n}$ so that the number-theoretic transform (NTT) can be realized efficiently and that polynomials can be transferred in NTT encoding.

	R-LWE ECDSA
1 B	507.5 (1.7)
1 KiB	505.9 (2.1)
10 KiB	490.9 (0.9)
100 KiB	397.6 (1.4)

Table 6.3: OpenSSL/Apache TLS simultaneous connections. Average median values (standard deviation).

Public parameters		
Parameters: $q = 12289 < 2^{14}, n = 1024$		
Error distribution: ψ_{16}		
Alice		**Bob**
$seed \xleftarrow{\$} \{0,1\}^{256}$		
$a \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$		
$s, e \xleftarrow{\$} \psi_{16}^{n}$		$s', e', e'' \xleftarrow{\$} \psi_{16}^{n}$
$b \leftarrow as + e$	$\xrightarrow{(b,seed)}$	$a \leftarrow \text{Parse}(\text{SHAKE-128}(seed))$
		$u \leftarrow as' + e'$
		$v \leftarrow bs' + e''$
$v' \leftarrow us$	$\xleftarrow{(u,r)}$	$r \xleftarrow{\$} \text{HelpRec}(v)$
$\nu \leftarrow \text{Rec}(v', r)$		$\nu \leftarrow \text{Rec}(v, r)$
$\mu \leftarrow \text{SHA3-256}(\nu)$		$\mu \leftarrow \text{SHA3-256}(\nu)$

Figure 6.2: The NewHope protocol.

Noise distribution

Alkim et al. found it challenging to implement a discrete Gaussian sampler efficiently *and* protected against timing attacks. Therefore, they changed the distribution of the LWE secret and error and replaced discrete Gaussians by the centered binomial distribution ψ_k of parameter $k = 16$. Sampling from the centered binomial distribution is easy and does not require high-precision computations or large tables. One simply samples from ψ_k by computing $\sum_{i=0}^{k} b_i - b'_i$, where the $b_i, b'_i \in \{0,1\}$ are uniform independent bits. The distribution ψ_k is centered, has variance $k/2$ and for $k = 16$ gives standard deviation $\varsigma = \sqrt{16/2}$.

Reconciliation

The reconciliation mechanism is generalized using an analog error-correction approach. One have $n = 1024$ coefficients to encode data into, but only want to transmit a 256-bit key. Therefore, one encodes one key bit into four coefficients. This gives increased error resilience which in turn allows for larger noise for better security.

	Normal	Optimized
Server key generation	258246	88920
Client key generation and shared key	384994	110986
Server shared key	86280	19422

Table 6.4: NewHope performance in cycles.

6.2.2 Performance

This system also aims at the 128-bit security level, but some of the parameters are changed:

- $n = 1024$
- $q = 12289$

The NewHope system uses a new reconcilation, so error distribution is not expressed with σ. Discrete Gaussians are replaced by the centered binomial distribution ψ_k of parameter $k = 16$.

Test runs were done on an Intel Core i7-4770K running at 3491.953 MHz. The implementation was written in the C programming language and compiled with `gcc-4.9.2` and flags `-O3 -fomit-frame-pointer -march=corei7-avx -msse2avx`. The running times are presented as the cycle count and are the median values over 1000 runs. The server key generation was measured to be 258246 cycles. The clients key generation and creation of shared key was measured to be 384994 cycles. The servers shared key creation was measured to be 86280 cycles.

Alim et al. also made an optimized AVX implementation. Newer Intel processors support Advanced Vector Extensions (AVX) that operate on vectors of 8 single-precision or 4 double-precision floating-point values in parallel. This makes it possible to optimize implementations targeting these processors. This optimized AVX implementation was compiled with `clang-3.5` and flags `-O3 -fomit-frame-pointer -march=native`. The server key generation was measured to be 88920 cycles. The clients key generation and shared key creation was measured to be 110986 cycles. The servers shared key creation was measured to be 19422 cycles. So the optimized implementation have a substantial speed up compared to the normal implementation, but it requires a new Intel processor.

Chapter 7

Practical Key Exchange

Another implementation of a ring-LWE based cryptosystem is the one presented by Vikram Singh in his paper *A Practical Key Exchange for the Internet using Lattice Cryptography* [29]. This key exchange is also based on the protocols presented by Peikert, but focus on the simpler case of cyclotomic rings whose degree is a power of two. Singh later relased another paper together with Arjun Chopra called *Even More Practical Key Exchanges for the Internet using Lattice Cryptography*, where they focus on the case of cyclotomic rings with degree $p-1$ for prime p [30]. This chapter will describe the cryptosystems presented in these two papers.

7.1 Practical Key Exchange

The first lattice-based cryptosystem presented by Singh focused on a simpler case than what Peikert proposed in his paper, namely cyclotomic rings whose degree is a power of two. This restriction also restrict the security levels of the scheme, but Singh claims that this hides complexities of ring arithmetic while still providing a reasonable diversity of practical security levels. The information in this section is from Singh's first paper on practical key exchange [29].

7.1.1 Protocol

Singh's implementation of the ring-LWE based cryptosystem is very similar to Peikert's description. As mentioned, this version focuses on cyclotomic

Alice		**Bob**
$pk = b = a \cdot s_1 + s_0$	$\xrightarrow{b}$	$pk = u = e_0 \cdot a + e_1$
		$SSV = v = e_0 \cdot b + e_2 =$
		$e_0 \cdot a \cdot s_1 + e_0 \cdot s_0 + e_2$
		key $= \lfloor v \rceil_2$
$w = u \cdot s_1$	$\xleftarrow{(u, \langle v \rangle_2)}$	mask $= \langle v \rangle_2$
key $=$ rec$(w, \langle v \rangle_2)$		

Figure 7.1: Basic key exchange algorithm for Practical Key Exchange.

rings with degree a power of two ($m = 2^\ell$ is a power of two). The basic key exchange algorithm can be seen in figure 7.1.1. *key* and *mask* are just other names of the shared key and the masking bits, which is denoted by μ and v' respectively by Peikert. Singh has introduced some changes from Peikert in the masking and reconciliation functions. The changes will be described briefly in this section.

Randomized Rounding

We recall from chapter 5 that Peikert introduced a randomized function dbl to temporarily scale cases with odd q to cases with even q and using the reconiliation mechanism on the even case instead. Singh has a more efficient way of solving this. Instead of scaling up, Singh introduces a simple randomized rounding procedure. The randomized rounding is more efficient than Peikert's up-scaling because it only needs to do a coin flip in a pair of edge cases. The details of the randomized rounding procedure can be found in Singh's paper [29].

7.1.2 Performance

Singh provides parameter choices for two levels of security: 128-bit (regular security) and 256-bit (high security). The parameters are as following:

128-bit

- $n = 512$
- $q = 25601$

- public key size = 7680 bits

256-bit

- $n = 1024$
- $q = 40961$
- public key size = 16384 bits

The key encapsulation method has been implemented in the C programming language and it has been tested on an Intel Core i5 4300U running at 1.9GHz with Turbo Boost increasing it to 2.9GHz. The code was compiled using `gcc 4.9.2` and flag `-O3`. For 128-bit security, the KEM1.Generate method used 135200 cycles on average, which is 54 microseconds. The KEM1.Encapsulate method used 222100 cycles on average, which is 89 microseconds. KEM1.Decapsulate used 52800 cycles on average, which is 21 microseconds. This totals to 410200 cycles, 164 microseconds. For 256-bit security, the KEM1.Generate method used 280600 cycles on average, which is 112 microseconds. The KEM1.Encapsulate used 457600 cycles on average, which is 183 microseconds. KEM1.Decapsulate used 113900 cycles on average, which is 340 microseconds. This totals to 852200 cycles, 340 microseconds.

7.2 Even More Practical Key Exchange

In *Even More Practical Key Exchanges for the Internet using Lattice Cryptography*, Singh and Chopra focus on the case of cyclotomic rings with degree

	128-bit		256-bit	
	Cycles	Microseconds	Cycles	Microseconds
KEM1.Generate	135200	54	280600	112
KEM1.Encapsulate	222100	89	457600	183
KEM1.Decapsulate	52800	21	113900	340
Total	410200	164	852200	340

Table 7.1: Performance of Practical Key Exchange running on an Intel Core i5 4300U 1.9 GHz (Turbo Boost to 2.9 GHz).

	m	n	q	σ	Security	Public Key Size
1	337	336	32353	3.192	96	5040 bits
2	433	432	35507	3.192	128	6912 bits
3	541	540	41117	3.192	160	8640 bits
4	631	630	44171	3.192	192	10080 bits
5	739	738	47297	3.192	224	11808 bits
6	821	820	49261	3.192	256	13120 bits

Table 7.2: Parameter sets of Even More Practical Key Exchange.

$p - 1$ for prime p. This allows for a greater degree of flexibility in choosing lattice dimension, which determines the security level and efficiency of the scheme [30].

7.2.1 Performance

As mentioned, having cyclotomic rings with degree $p-1$ for prime p allows for more flexibility of choosing security level and efficiency. Singh and Chopra presents six different sets of parameters which all have different security levels. Peikerts security analysis of ring-LWE gives a practical bound on the size of the modulus of $q \approx n^{3/2}$. To maintain consistency with the proof, Singh and Chopra also choose $q \equiv 1 \pmod{m}$. For a given m, the smallest $q > n^{3/2}$ that is congruent to 1 modulo m and provides a decryption failure rate of at most 2^{-80} is chosen. The pairs (m, q) with the smallest public keys are then selected. The sets of parameters are listed in table 7.3.

Singh and Chopra also ran some performance tests of their system. The tests was run on an 1.9 GHz Intel Core i5 4300U with Turbo Boost, increasing the clock rate to 2.9 GHz. The total runtime of the passively key exchange KEM1 was measured. The total runtime is the sum of the time for KEM1.Generate, KEM1.Encapsulate, and KEM1.Decapsulate. Both uniform sampling and Gaussian sampling was tested. The total runtime for parameter set 1 was measured to be 1401600 average cycle count for uniform sampling and 2703400 average cycle count for Gaussian sampling. Parameter set 2 was 1411600 for uniform sampling and 2982000 for Gaussian sampling. Parameter set 3 was 2952700 for uniform sampling and 4920600 for Gaussian sampling. Parameter set 4 was 3208500 for uniform sampling and 5331800 for Gaussian sampling. Parameter set 5 was 3032600 for uniform sampling and 6146600 for Gaussian sampling. Parameter set 6 was 3065300 for uniform

	Uniform	Gaussian
1	1401600	2703400
2	1411600	2982000
3	2952700	4920600
4	3208500	5331800
5	3032600	6146600
6	3065300	6071100

Table 7.3: Average cycle count of KEM1 running on Intel Core i5 4300U.

sampling and 6071100 for Gaussian sampling.

Chapter 8

Comparisons

After presenting some lattice-based cryptosystems individually, it is a good idea to look at comparable aspects of the schemes and put it together in a comparison. This chapter will take parameters, performance, etc., of the presented schemes and compare them to each other and also to the standard systems used today.

8.1 Comparisons of the presented schemes

8.1.1 Parameters

The parameters that are natural to compare when it comes to lattice-based cryptosystems, is the dimension of the lattice n and the prime q. The parameters of the ring-LWE based cryptosystems can be seen in table 8.1. The NTRU system is not included because it is constructed differently than the ring-LWE based cryptosystems.

As we can see from the table, there is not much difference on the choice of n. Both Post-quantum KE and NewHope are targeting the 128-bit security level and have $n = 1024$, while for the Practical Key Exchange, Singh claims that having $n = 512$ will provide at least 128-bit security. Having $n = 1024$ in the Practical Key Exchange will provide over 256-bit security. The move from cyclotomic rings with degree of power of two to cyclotomic rings with degree $p - 1$ for prime p allows for more flexibility in choosing parameters, and we can see that n in Even More Practical Key Exhcange is a bit lower than the other systems. When it comes to the prime q, it is much bigger in

	n	q
Post-quantum KE for TLS	1024	$2^{32} - 1$
NewHope	1024	12289
Practical KE 128	512	25601
Practical KE 256	1024	40961
Even More Practical KE 96	336	32353
Even More Practical KE 128	432	35507
Even More Practical KE 160	540	41117
Even More Practical KE 192	630	44171
Even More Practical KE 224	738	47297
Even More Practical KE 256	820	49261

Table 8.1: Parameters of the ring-LWE-based cryptosystems.

the Post-quantum Key Exchange for TLS than the other systems. Alkim et al. claims that the analysis of the failure probability from Bos et al. was far from tight, resulting in a too large q [3].

8.1.2 Key sizes

Key sizes are very important for a cryptographic scheme. Having big keys restricts the platforms the scheme can be implemented on, and also impacts the performance. Having smaller keys was one of the reasons for moving to elliptic curve cryptography.

Table 8.2 shows the public key sizes of NTRU [1], Practical Key Exchange [29], and Even More Practical Key Exchange [30] in bits. These values are the ones presented by the authors behind the schemes, and we can see that the key sizes are not that different, but NTRU reports the smallest key sizes. Table 8.3 shows the actual size of the communication between the two parties A and B doing a key exchange. These values are acquired from running the benchmark tool with the command `test_kex --bench` from the Open Quantum Safe (OQS) project [23]. The values are in bytes and were measured on an Intel Core i3-4030U running at 1.9GHz. NTRU does have the smallest communication size, which fits with the reported key sizes. Post-quantum Key Exchange have to transfer over double the data of NewHope, which probably is because of the large difference in the choice of q as seen in the previous section.

	Public key size (bits)
NTRU 112	5951
NTRU 128	6743
NTRU 192	9759
NTRU 256	12881
Practical KE 128	7680
Practical KE 256	16384
Even More Practical KE 96	5040
Even More Practical KE 128	6912
Even More Practical KE 160	8640
Even More Practical KE 192	10080
Even More Practical KE 224	11808
Even More Practical KE 256	13120

Table 8.2: Public key sizes of the lattice-based cryptosystems.

Scheme	**A $\rightarrow$ B**	**B $\rightarrow$ A**	**Total**
Post-quantum KE	4096	4224	8320
NewHope	1824	2048	3872
NTRU	1027	1022	2049

Table 8.3: Communication size (bytes) of Post-quantum KE, NewHope, and NTRU.

	128-bit	256-bit
Practical KE	410200	852200
Even More Practical KE (uniform)	1411600	3065300
Even More Practical KE (gaussian)	2982000	6071100

Table 8.4: Average total cycle count of KEM1 running on Intel Core i5 4300U.

Scheme	**Operation**	**Time (μs): mean (pop. stdev)**	**CPU cycles: mean (pop. stdev)**
Post-quantum KE	Alice 0	2154.598 (19.193)	4084122 (36275)
	Bob	3450.572 (33.900)	6540887 (64258)
	Alice 1	436.528 (3.010)	827377 (5807)
NewHope	Alice 0	143.069 (24.974)	271103 (47291)
	Bob	218.227 (34.187)	413575 (64728)
	Alice 1	37.456 (6.083)	70914 (11481)
NTRU	Alice 0	2783.828 (304.947)	5276705 (577701)
	Bob	288.917 (62.924)	547523 (119212)
	Alice 1	175.666 (26.846)	332891 (50827)

Table 8.5: Speed measurements of Post-quantum KE, NewHope, and NTRU.

8.1.3 Speed

Comparing the run times of the different schemes is not that easy. The measurements presented in the earlier chapters are not measured the same way and on the same hardware. The only measurements that are reasonable to compare, are those of *Practical Key Exchange* and *Even More Practical Key Exchange*. Both was measured running on an Intel Core i5 4300U, and can be seen in table 8.4. We can see that the move from cyclotomic rings whose degree is a power of two to cyclotomic rings with degree $p-1$ for prime p, have a big impact on the performance of the scheme.

Information in table 8.5 is aquired from running the benchmark tool from OQS. The tests were run on an Intel Core i3 4030U at 1.90 GHz, and simulate a key exchange between Alice and Bob where Alice is the instigator. Out of Post-quantum Key Exchange, NewHope, and NTRU, NewHope is the fastest by far with a total of 398.752 μs and 755592 cycles. NTRU takes 3248.411 μs and uses 6157119 cycles. A lot more than NewHope, but it is only the initial

Security Level	NTRU Key Size	ECC Key Size	RSA Key Size
112	5951	224	2048
128	6743	256	4096
192	9757	384	7680
256	12881	512	15360

Table 8.6: Public key sizes (bits) of NTRU, ECC, and RSA.

operation for Alice that is really slow. The other operations are almost as fast as NewHope. For Post-quantum Key Exchange, all operations are slow. The total time is 6041.698 μs and it uses 11452386 cycles. Having a much bigger q seems to slow it down considerably compared to the other schemes.

8.2 Comparisons with today's standards

If you want to replace today's standard cryptoschemes, the new schemes should ideally perform better than the ones replaced or atleast perform the same. The two most common algorithms to exchange or agree on a secret key is RSA and Diffie-Hellman, so it is natural to compare the lattice-based schemes to those.

NTRU is the scheme of those presented in this thesis that has the smallest key sizes. Table 8.6 compares it with RSA and Elliptic Curve Cryptography (ECC). The values are from the NTRU GitHub-page [1]. For the lower security levels, NTRU has the largest keys by quite a lot, but it scales better with the security levels as RSA have the largest key at 256-bit security. ECC have much smaller keys than both RSA and NTRU. This is also one of the reasons why ECC is used in favor of ordinary RSA, as the small key size allows for the cryptoscheme to run on smaller platforms and have better performance. This can be seen in table 8.7. ECC performs much better than RSA, which is really struggling with the large keys. NTRU does not have that problem. Even though the keys are very large compared with ECC, NTRU outperforms ECC by a large margin. It is almost six times faster on the 256-bit security level.

Security Level	**NTRU Ops/Sec**	**ECC Ops/Sec**	**RSA Ops/Sec**
112	2284	951	156
128	1896	650	12
192	1034	285	8
256	638	116	1

Table 8.7: Performance of NTRU, ECC, and RSA.

Chapter 9

Summary and further work

With quantum computers seeming like a possibility in the future, the standard public-key cryptosystems used today are vulnerable and new, quantum-resistant systems are needed. There are several different types of cryptosystems that could be possible replacements, for example hash-based, code-based, and lattice-based cryptosystems. This thesis focused on lattice-based cryptosystems, and presented a few of them. The lattice-based cryptosystems are of two types: NTRU and ring-LWE. Both NTRU and ring-LWE are operating on polynomial rings, and have strong, provable security against quantum computers.

The lattice-based cryptosystems perform well. All the cryptosystems presented in this thesis outperform the standard systems used today. Key sizes are a bit larger on the lower security levels, but scale better when increasing the security level. All the lattice-based cryptosystems have larger keys than elliptic curve cryptography, so they might not be suitable for small devices with limited storage space.

When compared to RSA and Diffie-Hellman, lattice-based cryptosystems are much more complicated. Lattices are more complicated than integers, so encryption and decryption are in turn more complicated operations. This makes it harder for programmers with little to no mathematical background to understand and use the cryptosystems, which can lead to systems with faulty security.

If it is possible to make the lattice-based cryptosystems a bit more accessible for the general public, these systems are good candidates for new quantum-resistant standards. They are fast and have strong, provable security against quantum algorithms.

9.1 Further work

This section will present possible further work that this thesis did not cover.

- Implementing a lattice-based cryptosystem. Doing an implementation requires full understanding of the theory of the systems, and can uncover possible errors that can occur when people with little to no mathematical background tries to implement such complicated cryptosystems.

- More thorough performance testing. When presenting a new cryptosystem, authors often include some performance numbers. A problem is that different schemes are tested in different ways on different hardware, making it hard to compare the test results. Ideally, all cryptosystems should be tested in the same way on the same hardware. The Open Quantum Safe project is working towards this, and have gathered several different implementations of lattice-based cryptosystems in a testing suite, which was used in this thesis. However, there are systems that are yet to be added, so improvements are possible. The Open Quantum Safe project is an open-source project, so anyone can contribute.

- As already mentioned, lattice-based cryptosystems are much more advanced than RSA and Diffie-Hellman. Making them easier to understand is a hard task, but one that is necessary for a widespread adoption of lattice-based cryptography.

Bibliography

[1] URL: https://github.com/NTRUOpenSourceProject/ntru-crypto (visited on 06/05/2016).

[2] National Security Agency. *NSA Suite B Cryptography*. 2015. URL: https://www.nsa.gov/ia/programs/suiteb_cryptography/ (visited on 04/12/2016).

[3] Erdem Alkim et al. *Post-quantum key exchange - a new hope*. Cryptology ePrint Archive, Report 2015/1092. http://eprint.iacr.org/2015/1092. 2015.

[4] Joppe W. Bos et al. *Post-quantum key exchange for the TLS protocol from the ring learning with errors problem*. Cryptology ePrint Archive, Report 2014/599. http://eprint.iacr.org/2014/599. 2014.

[5] Ran Canetti and Hugo Krawczyk. *Security Analysis of IKE's Signature-based Key-Exchange Protocol*. Cryptology ePrint Archive, Report 2002/120. http://eprint.iacr.org/2002/120. 2002.

[6] Dong Pyo Chi et al. *Lattice Based Cryptography for Beginners*. Cryptology ePrint Archive, Report 2015/938. http://eprint.iacr.org/2015/938. 2015.

[7] W. Diffie and M. Hellman. "New Directions in Cryptography." In: *IEEE Trans. Inf. Theor.* 22.6 (Sept. 2006), pp. 644–654. ISSN: 0018-9448. DOI: 10.1109/TIT.1976.1055638. URL: http://dx.doi.org/10.1109/TIT.1976.1055638.

[8] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes." In: *Advances in Cryptology — CRYPTO' 99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings*. Ed. by Michael Wiener. Berlin, Heidelberg: Springer Berlin Heidelberg,

1999, pp. 537–554. ISBN: 978-3-540-48405-9. DOI: 10.1007/3-540-48405-1_34. URL: http://dx.doi.org/10.1007/3-540-48405-1_34.

[9] O. Goldreich et al. “Approximating Shortest Lattice Vectors is Not Harder Than Approximating Closet Lattice Vectors.” In: *Inf. Process. Lett.* 71.2 (July 1999), pp. 55–61. ISSN: 0020-0190. DOI: 10.1016/S0020-0190(99)00083-6. URL: http://dx.doi.org/10.1016/S0020-0190(99)00083-6.

[10] Lov K. Grover. “A Fast quantum mechanical algorithm for database search.” In: (1996). arXiv: quant-ph/9605043 [quant-ph].

[11] Sean Hallgren and Ulrich Vollmer. “Post-Quantum Cryptography.” In: ed. by Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Chap. Quantum computing, pp. 15–34. ISBN: 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_2. URL: http://dx.doi.org/10.1007/978-3-540-88702-7_2.

[12] Matthew Hayward. *Quantum Computing and Shor's algorithm*. Revised in 2015. 1999.

[13] Jens Hermans, Frederik Vercauteren, and Bart Preneel. “Topics in Cryptology - CT-RSA 2010: The Cryptographers' Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings.” In: ed. by Josef Pieprzyk. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. Chap. Speed Records for NTRU, pp. 73–88. ISBN: 978-3-642-11925-5. DOI: 10.1007/978-3-642-11925-5_6. URL: http://dx.doi.org/10.1007/978-3-642-11925-5_6.

[14] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *NTRU: A Public Key Cryptosystem*. 1999.

[15] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *NTRU: A Ring-Based Public Key Cryptosystem*. 1998.

[16] European Telecommunications Standards Institute. *Quantum Safe Cryptography and Security*. 2015.

[17] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. “On Ideal Lattices and Learning with Errors over Rings.” In: *Advances in Cryptology – EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 – June 3, 2010. Proceedings*. Ed. by Henri Gilbert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 1–23. ISBN:

978-3-642-13190-5. DOI: 10.1007/978-3-642-13190-5_1. URL: http://dx.doi.org/10.1007/978-3-642-13190-5_1.

[18] Daniele Micciancio and Oded Regev. “Post-Quantum Cryptography.” In: ed. by Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. Chap. Lattice-based Cryptography, pp. 147–191. ISBN: 978-3-540-88702-7. DOI: 10.1007/978-3-540-88702-7_5. URL: http://dx.doi.org/10.1007/978-3-540-88702-7_5.

[19] Ashley Montanaro. “Quantum algorithms: an overview.” In: *Npj Quantum Information* 2 (Jan. 2016). Review Article, 15023 EP -. URL: http://dx.doi.org/10.1038/npjqi.2015.23.

[20] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000. ISBN: 0-521-63503-9.

[21] Chris Peikert. *A Decade of Lattice Cryptography*. Cryptology ePrint Archive, Report 2015/939. http://eprint.iacr.org/. 2015.

[22] Chris Peikert. *Lattice Cryptography for the Internet*. Cryptology ePrint Archive, Report 2014/070. http://eprint.iacr.org/2014/070. 2014.

[23] The Open Quantum Safe Project. *Open Quantum Safe*. 2016. URL: https://openquantumsafe.org/ (visited on 06/13/2017).

[24] R. L. Rivest, A. Shamir, and L. Adleman. “A Method for Obtaining Digital Signatures and Public-key Cryptosystems.” In: *Commun. ACM* 21.2 (Feb. 1978), pp. 120–126. ISSN: 0001-0782. DOI: 10.1145/359340.359342. URL: http://doi.acm.org/10.1145/359340.359342.

[25] Inc. Security Innocation. *NTRU PKCS Tutorial*. 2014.

[26] National Security Agency/Central Security Service. *Commerical National Security Algorithm Suite and Quantum Computing FAQ*. 2016.

[27] Peter W. Shor. “Algorithms for quantum computation: discrete logarithms and factoring.” In: *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*. Nov. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[28] Joseph H. Silverman. *NTRU and Lattice-Based Crypto: Past, Present, and Future*. 2015.

[29] Vikram Singh. *A Practical Key Exchange for the Internet using Lattice Cryptography*. Cryptology ePrint Archive, Report 2015/138. http://eprint.iacr.org/2015/138. 2015.

[30] Vikram Singh and Arjun Chopra. *Even More Practical Key Exchanges for the Internet using Lattice Cryptography*. Cryptology ePrint Archive, Report 2015/1120. http://eprint.iacr.org/2015/1120. 2015.

[31] Michael Weber. *GoldBug - Encrypted Communications*. URL: https://sourceforge.net/projects/goldbug/ (visited on 07/19/2017).