BACHELOR

NTRU+
Analyzing variants, attacks and security of a lattice-based post-quantum scheme

Steenbakkers, Luc G.A.M.

Award date:
2023

NTRU+

Analyzing variants, attacks and security of a lattice-based post-quantum scheme

Bachelor's Thesis

L.G.A.M. Steenbakkers
1599682

Supervisor:
Prof. dr. Tanja Lange

Daily Supervisor:
Jolijn Cottaar, MSc

Final version

Eindhoven, August 2023

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Coding Theory and Cryptology

Contents

1 Introduction

Nowadays, digital communication is indispensable in our daily lives. Businesses use different forms of digital communication for both internal communication between employees and external communication with customers or partners around the world. Moreover, education and online services such as video-lectures, e-learning or video calls all rely on digital communication. Secret services, government organizations and emergency services can use digital communication to reach out to citizens or communicate within the organization.

This communication must be secure to protect sensitive information, prevent tampering, and maintain trust in the communication channel. To achieve these safe communication channels, encryption algorithms exist to ensure confidentiality, integrity and authenticity of data. However, in this thesis, we will only focus on achieving confidentiality of communication through encryption algorithms. At the core of any such encryption algorithm is some mathematical problem, which can be computed efficiently in one way but is extremely difficult to reverse. A well known Public Key Encryption (PKE) algorithm was proposed by Rivest, Shamir and Adleman in 1978 [31], and is based around the mathematical problem of integer factorization. This problem describes how it is easy to multiply two large prime numbers, but very difficult to factor a large number into two primes. Another scheme that is often used is the Diffie-Hellman key agreement protocol [11], which is built around the Discrete Logarithm Problem (DLP). The DLP suggests that it is difficult to find a number $a \in \mathbb{Z}$ for which $g^a = A \mod p$, where p is a known prime, and g and A are known integers.

However, with the rise of quantum computers, these algorithms might not be secure in the near future. If scientist ever manage to construct a quantum computer with sufficiently many quantum bits to perform calculations with large numbers, then algorithms exist to efficiently reverse some of the difficult mathematical problems that we base our encryption on. For example, an algorithm already discovered by Peter Shor in 1994, known as Shor's algorithm [36], can be implemented to find the prime factors of an integer or solve a DLP in polynomial time.

To counter this, Post-Quantum Cryptography (PQC) schemes are designed around problems for which no efficient quantum algorithm exists yet. To find such schemes, the National Institute of Standards and Technologies (NIST) of the United States Department of Commerce organized the Post-Quantum Cryptography Standardization search, where participants proposed quantum-secure digital signature schemes, key exchange algorithms and PKE schemes. This competition consists of multiple rounds, and during each round cryptanalysts from around the world have the opportunity to test the robustness of the submitted systems, after which the creators of the schemes can choose to alter their systems to improve security. Based on the attacks and vulnerabilities that are found, the organizers then announce which schemes are advancing to the next round.

Another search, with a similar goal as the NIST, is the Korean Post-Quantum Competition (KPQC). The first round of this competition takes place from November 2022 to November 2023, and NTRU+ is one of the PKE submissions. It is based on NTRUEncrypt, a lattice-based encryption method that was introduced by Hoffstein, Pipher and Silverman in 1998 [18]. However, NTRUEncrypt, like all PKE systems, contains a trade-off between speed and security. One of the main issues with NTRUEncrypt is the possibility for decryption failures: if the ciphertext is created incorrectly or bit flips occur during transmission, then NTRUEncrypt will not be able to decrypt it. The aim of NTRU+ is to reduce the probability of this happening as much as possible, using an Average Case to Worst Case correctness error transform. Moreover, it aims to speed up calculations by choosing a ring other than the NTRUEncrypt ring, on which they apply the Number Theoretic Transform (NTT). Thus, their system supports faster arithmetic while also offering more security.

The aim of this project is to research lattices and lattice based cryptography, find differences and similarities between NTRU+ and other NTRU instantiations, and research attacks against other NTRU variants and whether they would be effective against NTRU+. We will start by discussing some preliminaries, which will be helpful later on. Then, the original NTRUEncrypt will be discussed, along with the third round NIST submission NTRU Prime. Moreover, the NTRU variant NTTRU, which utilizes the Number Theoretic Transform like NTRU+, will be discussed. Then, we will go over the inner workings of NTRU+. It should be mentioned that different versions of the NTRU+ paper have been uploaded overtime, without informing possible reviewers. For consistency, the content of this thesis is based on the very first NTRU submission from the 21st of November 2022.

After studying the NTRU+ system, several attacks on NTRU variants will be discussed, along with reasoning why they will or will not be effective against NTRU+. At last, we will briefly touch upon some attack vectors that have not been discussed in detail. With these attack vectors, we will suggest some opportunities for future work.

2 Preliminaries

This section contains the mathematical background needed to comprehend the content of this thesis. First, we will quickly touch upon cryptographic hash functions. Then, we will explain how Public Key Encryption works, and what a Key Encapsulation Mechanism is. Next, the mathematical definitions of groups, rings and fields will be discussed, along with some properties and implementations that will prove to be useful later on. Finally, we will discuss the basics of lattices and a useful algorithm that can be used in attacks later on.

2.1 Cryptographic Hash Functions

Often in cryptography, we require a function that maps a string of data to a pseudorandom string of a fixed number of bits. Functions which achieve this are called Cryptographic Hash Functions (CHF). Let $\{0,1\}^n$ be the set of all bitstrings of length n, and let $\{0,1\}^*$ be the set of all bitstrings of arbitrary length. Then, we can define a Cryptographic Hash Function as follows:

Definition 2.1 (Cryptographic Hash Function)**.** A function defined as

$$H : \{0,1\}^* \to \{0,1\}^n$$

is a Cryptographic Hash Function when

- the probability of outputting a specific n-bit string, for a random input, is 2^{-n}.
- it is preimage resistant, meaning that it is computationally difficult, given a $y \in \{0,1\}^n$ to find an $x \in \{0,1\}^*$ such that $H(x) = y$.
- it is second preimage resistant, meaning that it is computationally difficult, given a $x \in \{0,1\}^*$ with $H(x) = y$, to find $x' \in \{0,1\}^*$, with $x \neq x'$, such that $H(x') = y$ as well.
- it is collision resistant, meaning that it is computationally difficult to find $x, x' \in \{0,1\}^*$, with $x \neq x'$, such that $H(x) = H(x')$.

Moreover, a characteristic of a CHF is the diffusion property, which states that a change in just one bit of the input of the hash function, should change every output bit with a probability of $\frac{1}{2}$.

With these properties, we note that any message m can be represented by its hash $H(m)$. Then, this hash can be used publicly while the underlying message m is still secret. This consequence will be useful later on for defining a Key Encapsulation Mechanism.

2.2 Public Key Encryption

Public Key Encryption (PKE) or asymmetric encryption is a type of encryption algorithm which has separate, distinct keys for encryption and decryption. Key generation creates a keypair in some sets or keyspaces $(\mathcal{K}_1, \mathcal{K}_2)$, consisting of a public key $pk \in \mathcal{K}_1$ and a private or secret key $sk \in \mathcal{K}_2$:

$$\begin{aligned} \text{Gen} : (\cdot) &\to (\mathcal{K}_1, \mathcal{K}_2) \\ (\cdot) &\mapsto (pk, sk) \end{aligned}$$

where $(\cdot)$ denotes some security parameter depending on the PKE algorithm. Now, for a message space $\mathcal{M}$ and for some message $m \in \mathcal{M}$, this message can be encrypted to form a ciphertext $c \in \mathcal{C}$, where $\mathcal{C}$ is the set of ciphertexts, under the encryption function

$$\begin{aligned} \text{Enc} : (\mathcal{K}_1, \mathcal{M}) &\to \mathcal{C} \\ (pk, m) &\mapsto c. \end{aligned}$$

Similarly, c can be decrypted as

$$\begin{aligned}\mathsf{Dec} : (\mathcal{K}_2, \mathcal{C}) &\to \mathcal{M} \\ (sk, c) &\mapsto m.\end{aligned}$$

Now, two parties, hereafter denoted as Alice and Bob, can communicate by sharing their public keys. Alice generates keypair (pk_A, sk_A) and shares pk_A with Bob. Bob encrypts a message $m \in \mathcal{M}$ as

$$c = \mathsf{Enc}(pk_A, m),$$

and sends c to Alice. Now, Alice recovers the message m by calculating

$$m = \mathsf{Dec}(sk_A, c).$$

But if only Alice generates a keypair, it means that Alice can only receive messages. Hence, if Alice and Bob want to communicate, Bob will also generate a keypair (pk_B, sk_B) and send his public key to Alice. Then, as shown in Figure 1, Bob and Alice can both encrypt messages for each other. Note that any third party is not able to decrypt the ciphertexts, as they only know pk_A and pk_B.

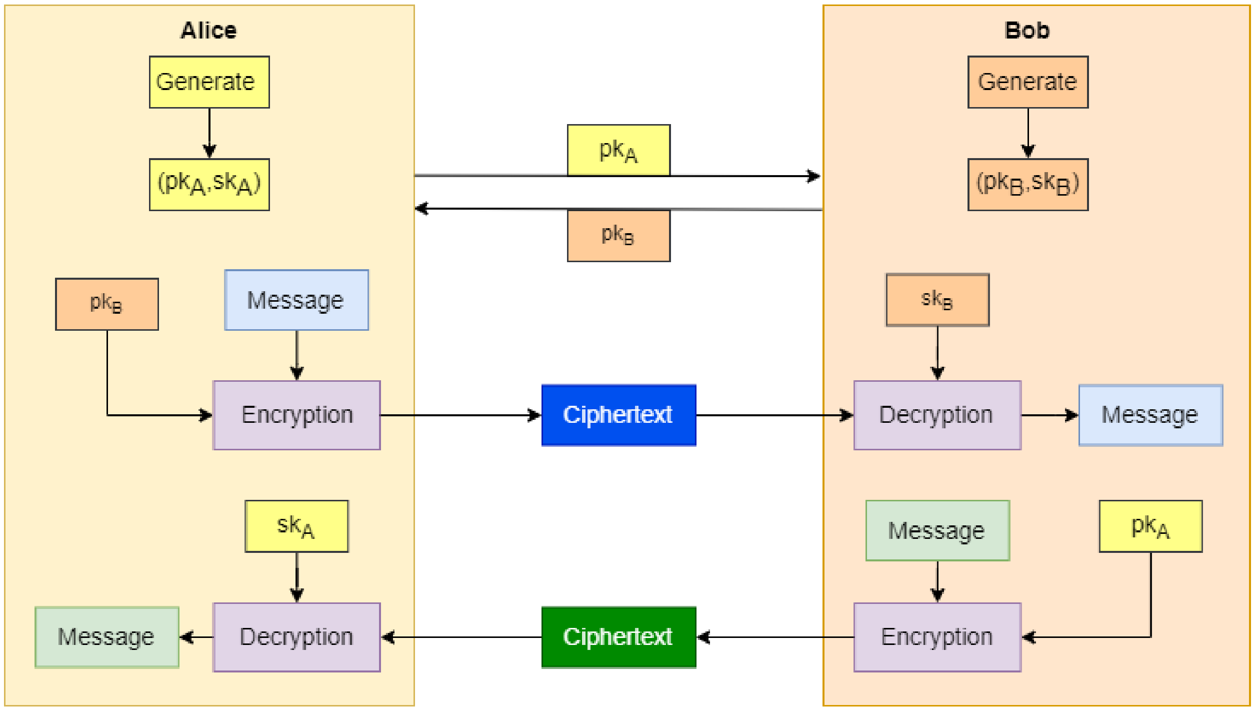

Figure 1: Illustration of a PKE scheme

The main drawback of asymmetric encryption is the time it takes to encrypt and decrypt a message, as these algorithms are often complicated and use large parameters to increase security.

2.2.1 IND-CPA security

The weakest notion of security is “indistinguishability under chosen plaintext attack” security, or IND-CPA security. To test whether a cryptographic scheme is IND-CPA secure, the IND-CPA security game exists. In this game, for some PKE $=$ (Gen, Enc, Dec), an adversary is given a public key pk and allowed to generate two valid messages $m_0, m_1 \in \mathcal{M}$, after which the challenger chooses one of them to encrypt using $\mathsf{Enc}(pk, m_i) = c_i$,

$i \in \{0,1\}$. Then, the adversary tries to guess which of the messages was encrypted. We define the advantage for the adversary as

$$\left|\mathbb{P}(m_{\text{guess}} = m_i) - \frac{1}{2}\right|$$

If a PKE = (Gen, Enc, Dec) is IND-CPA secure, then this advantage should be sufficiently small. More formally, we say that a ciphertext is IND-CPA secure if the advantage for the adversary decreases exponentially as the keysize grows. Note that IND-CPA security requires that Enc is randomized as otherwise the adversary could just encrypt m_0 and m_1 and check which of the encryptions matches the challenge ciphertext, thus win every game.

2.3 Key Encapsulation Mechanism

The counterpart of asymmetric encryption is symmetric encryption, where the key for encryption and decryption is the same. Because symmetric encryption and decryption is often much faster than asymmetric algorithms, it is the preferred way of encryption for many instances. However, the problem with such a symmetric system is how to get the key to both the sender and receiver. Simply sending the key over not yet secured communication channels will nullify the result, as any third party can then also access the secret key.

A Key Encapsulation Mechanism (KEM) is a protocol that securely exchanges symmetric keys over an insecure channel. It requires three algorithms: one for keypair generation, one for encapsulation of the symmetric key and one for decapsulation of this symmetric key. Given a PKE it is easy to construct a KEM as follows: First, the key generation is done by Bob using a PKE algorithm, which has keys in the keyspaces $(\mathcal{K}_1, \mathcal{K}_2)$. Then, Alice uses the encapsulation function Enc with Bob's public key from the generated keypair, and uses it to encrypt a random message m sampled from the message space $\mathcal{M}$. She also calculates the hash $H(m)$, which will function as the symmetric secret key. After encryption, the message m becomes a ciphertext in the ciphertext space $\mathcal{C}$:

$$\begin{aligned} \textsf{Encap} : \mathcal{K}_1 &\to (\mathcal{C}, \mathcal{K}) \\ pk &\mapsto (c, k) \end{aligned}$$

which internally uses

$$\begin{aligned} m &\leftarrow \mathcal{M}, \\ c &= \textsf{Enc}(pk, m), \\ k &= H(m). \end{aligned}$$

Now this ciphertext c is sent to Bob, who decapsulates the symmetric key by using his private key to decrypt the ciphertext, yielding the message m again:

$$\begin{aligned} \textsf{Decap} : (\mathcal{K}_2, \mathcal{C}) &\to \mathcal{K} \\ (sk, c) &\mapsto k \end{aligned}$$

which internally uses

$$\begin{aligned} m &= \textsf{Dec}(sk, c), \\ k &= H(m). \end{aligned}$$

Now, Bob can also hash this message, such that both Alice and Bob share the secret symmetric key $H(m)$. This symmetric key can now be used with a fast Symmetric Key Encryption (SKE) algorithm, such as AES. After the initial asymmetric key exchange of the shared symmetric key, Alice and Bob can continue using this key to encrypt messages, so they will only have to do the relatively slow encapsulation and decapsulation once. In Figure 2, the asymmetric key exchange is shown in blue. After the successful exchange of the shared key, many more messages in green can be encrypted using this key, although only one is illustrated. These messages can be encrypted and decrypted by both Alice and Bob and therefore they can be sent both ways.

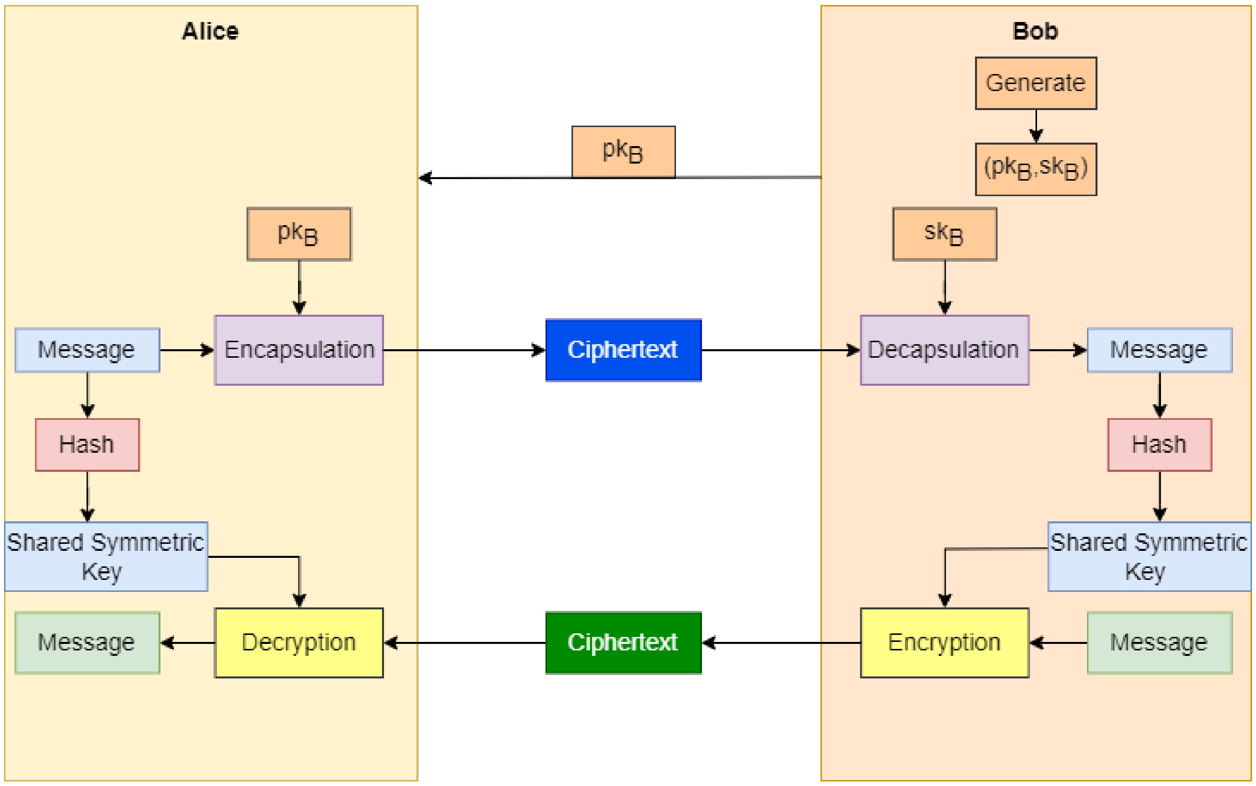

Figure 2: Visualization of a KEM and start of using symmetric encryption for sending messages

2.3.1 IND-CCA security

A strong notion of security is "indistinguishability under adaptive chosen-ciphertext attack" security, or IND-CCA security. To test whether a KEM ciphertext is IND-CCA secure, the IND-CCA game exists. Here, an adversary is given pk and is once again allowed to generate two plaintexts $m_0, m_1 \in \mathcal{M}$, after which the challenger randomly selects one m_i to encrypt it as Encap$(pk, m_i) = c_i$ for $i \in \{0, 1\}$. Hereafter, the challenger sends the ciphertext c_i to the adversary. The adversary again has to guess which message was encrypted. In this game the adversary is additionally permitted to request decryptions of ciphertexts under the secret key, except for requesting decryptions of the ciphertext c_i. Again, the advantage for the adversary in a KEM setting is denoted as

$$\left|\mathbb{P}(m_{\text{guess}} = m_i) - \frac{1}{2}\right|$$

If a KEM = (Gen, Encap, Decap) is IND-CCA secure, then this advantage is negligible.

2.4 Discrete mathematics

The NTRU+ system is a lattice-based cryptographic system, which relies heavily on rings, ideals and other algebraic concepts. These definitions will be introduced in the following paragraphs, alongside some examples. Note that the notions of groups, rings and fields are not always discrete by definition, but in this thesis we will only consider discrete versions of them. In this chapter, most of the definitions given are from the book "Abstract Algebra: Theory and Applications", written by Judson in 1994 [22].

2.4.1 Groups

We define a group as a set G with an associative operation $*$, with an identity element and an inverse. The operation must map two elements in the set to a third element, which also lies in the set G.

Definition 2.2 (Group)**.** A set G and an operation $* : G \times G \to G$ form a group with an inverse $^{-1} : G \to G$ and identity element e if

1. $*$ is associative, so $\forall a, b, c \in G$ it holds that
$$(a * b) * c = a * (b * c)$$
2. e is an identity element, i.e., $\forall a \in G$ it holds that
$$a * e = e * a = a$$
3. Every element in G has an inverse, thus $\forall a \in G$, there exists $a^{-1} \in G$ such that
$$a * a^{-1} = e = a^{-1} * a.$$

Some groups are created with the same operation, inverse and identity as another group, but only applying on a smaller set of elements. Such groups, which exist on a subset of an existing group, are called subgroups.

Definition 2.3 (Subgroup)**.** Let $(G, *, ^{-1}, e)$ be a group, and let $H \subseteq G$. Then H is a subgroup of G if $(H, *, ^{-1}, e)$ is a group under the same operation, inverse and identity element.

A further extension to the notion of a group is an abelian group, which is a group where the operation is commutative on G.

Definition 2.4 (Abelian Group)**.** A group $(G, *, ^{-1}, e)$ is called abelian if the operation $*$ is commutative on G, i.e., $\forall a, b \in G$, it holds that $a * b = b * a$.

To place these definitions in the context of this thesis, we will give examples of discrete groups. Note that a discrete group is simply a group where around each element of G, there exists a neighborhood not containing other elements of G.

Example 1. The set of integers $\mathbb{Z}$ forms a group under addition $+$, with inverse $^{-1} : a \to -a$ for $a \in G$ and identity 0. We can check this by verifying that

1 The operation $+$ is associative, as $(a + b) + c = a + b + c = a + (b + c)$.

2. For every $a \in G$ we have that $a + 0 = 0 + a = a$, hence 0 is the identity element in $\mathbb{Z}$.
3. The inverse $-a$ exists for every $a \in G$, and $a - a = 0 = -a + a$

2.4.2 Rings

A set with two binary operations is called a ring, if it adheres to some properties. Usually, these binary operations are called addition and multiplication.

Definition 2.5 (Ring)**.** A set R with two operations, $+ : R \to R$ and $* : R \to R$, forms a ring when it satisfies the ring axioms:

1. R forms an abelian group with $+$, $^{-1} : a \to -a$ and $e = \mathbf{0}$.
2. The operator $*$ is associative and R contains a multiplicative identity $\mathbf{1}$ such that $a * \mathbf{1} = a = \mathbf{1} * a$, $\forall a \in R$.
3. The operator $*$ is distributive with respect to $+$, so
$$a * (b + c) = a * b + a * c$$
$$(a + b) * c = a * c + b * c.$$

Now, we can extend Example 1 to form the set of all polynomials with integers coefficients under the usual operations of addition and multiplication.

Example 2. The set of polynomials with integers coefficients $\mathbb{Z}[x]$ forms a ring, as we expect them to work under addition $+$ and multiplication $\cdot$, which we can verify by checking:

1. $Z[x]$ forms an abelian group under addition, with the inverse function $a(x) \to -a(x)$ for $a(x) \in \mathbb{Z}[x]$ and identity 0. Here, $-a(x)$ denotes that every coefficient of a(x) is negated. The fact that $\mathbb{Z}[x]$ forms a group under addition follows directly from Example 1, because addition is applied coefficient wise and so is addition of elements in $\mathbb{Z}$. Addition is also commutative since $\forall a(x), b(x) \in \mathbb{Z}[x]$ it holds that $a(x) + b(x) = b(x) + a(x)$.
2. Multiplication $\cdot$ is associative as for all $a(x), b(x), c(x)$ we have $(a(x) \cdot b(x)) \cdot c(x) = a(x) \cdot b(x) \cdot c(x) = a(x) \cdot (b(x) \cdot c(x))$, and we have the multiplicative identity element $1 \in \mathbb{Z}[x]$ such that $a(x) \cdot 1 = a(x)$ for every $a(x) \in \mathbb{Z}[x]$.
3. Multiplication is distributive with respect to addition since $\forall a(x), b(x), c(x) \in \mathbb{Z}[x]$ we have $a(x) \cdot (b(x) + c(x)) = a(x) \cdot b(x) + a(x) \cdot c(x) = (a(x) + b(x)) \cdot c(x)$.

We have now seen what defines a ring, and how we can verify whether a set and its operations form a ring. Now, we will give a definition which gives a bit more structure to the definition of a ring.

Definition 2.6 (Commutative ring)**.** A ring R with operation $+$ and $*$ is called commutative when the operation $*$ is commutative on R, i.e., for all $a, b \in R$ it holds that $a * b = b * a$.

Finally, if we only consider commutative rings, we can give the definition of an ideal and start constructing quotient rings, which will be needed later on.

Definition 2.7 (Ideal)**.** Let R be a commutative ring and let I be a non-empty, additive subgroup of R. Then I is called an ideal of R if $\forall a \in I$, and $\forall r \in R$, it holds that $r * a \in I$.

Furthermore, an ideal I is called a proper ideal if it is different from R i.e., $R \neq I$ and thus the multiplicative identity $1 \notin I$. This is apparent since if $1 \in I$ then $r * 1 \in I \ \forall r \in R$, and then $R = I$.

We can build on Example 1 by taking the subgroup $p\mathbb{Z}$ where p is a prime. This construction will eventually be useful for NTRU as this forms a proper ideal that we can use to create a polynomial ring. However, first we show that this is indeed a proper ideal.

Example 3. The subgroup $p\mathbb{Z}$ of $\mathbb{Z}$ is a proper ideal of $(\mathbb{Z}, +, \cdot, a \to -a, 0, 1)$. First note that $(p\mathbb{Z}, +, a \to -a, 0)$ is a nonempty subgroup of $\mathbb{Z}$ under addition. Then $p\mathbb{Z}$ is an ideal since $\forall r \in \mathbb{Z}$ and $\forall a \in p\mathbb{Z}$, we can decompose $a = p \cdot n$ for $n \in \mathbb{Z}$ such that $r \cdot a = p \cdot r \cdot n \in p\mathbb{Z}$. Moreover, it is a proper ideal because $1 \notin p\mathbb{Z}$ but $1 \in \mathbb{Z}$.

From now on, we will consider the set of polynomials with integer coefficients $\mathbb{Z}[x]$. Note that, by extension of Example 3, $p\mathbb{Z}[x]$ is also a proper ideal of $\mathbb{Z}[x]$.

A useful property of proper ideals is that they can be used to create quotients rings: If we take a proper ideal I of a commutative ring R then R/I again forms a ring.

Theorem 1. *Let R be a commutative ring and let I be a proper ideal of R. Then R/I forms a quotient ring of R modulo I, under the usual operations of $+$ and $*$ in R.*

We can now construct a quotient ring combining Example 2 and Example 3.

Example 4. If we take the ring $\mathbb{Z}[x]$ and a corresponding proper ideal $p\mathbb{Z}[x]$ we can construct the ring $(\mathbb{Z}/p\mathbb{Z})[x]$ which can be represented as polynomials with integer coefficients in the set $\{0, 1, \ldots, p-1\}$. This ring uses the standard operations $+$ for addition and $\cdot$ for multiplication, with additive identity 0 and multiplicative identity 1.

Finally, since the above construction again forms a ring, we can search for a new proper ideal and construct a new quotient ring. Let us take the proper ideal generated by $x^n - 1$ where $n \in \mathbb{Z}$ of the ring $(\mathbb{Z}/p\mathbb{Z})[x]$. Then we can construct the ring

$$(\mathbb{Z}/p\mathbb{Z})\,[x]/(x^n-1).$$

Note that in this ring, $x^n - 1 = 0$ so $x^n = 1$, and therefore this is a polynomial ring where each term has exponent at most $n-1$. Now, multiplications by powers of x in this ring are simply shifts of the coefficients.

Example 5. Let us take $p = 13$ and $n = 7$ to obtain the ring

$$(\mathbb{Z}/13\mathbb{Z})\,[x]/(x^7-1).$$

Now take an element in this ring, for example $f(x) = 2x^6 + 7x^5 + 9x^4 + 5x^3 + 12x + 1$. Multiplying with x yields

$$\begin{aligned} x \cdot f(x) &= 2x^7 + 7x^6 + 9x^5 + 5x^4 + 12x^2 + x \\ &= 7x^6 + 9x^5 + 5x^4 + 12x^2 + x + 2. \end{aligned}$$

Note that each coefficient has been shifted to the left by 1. This product is called a *convolution*.

2.4.3 Fields

In later chapters, we will be working with rings where we do not just want all elements to have an inverse under addition, but also all non-zero elements to be invertible under multiplication. Such a ring is called a field:

Definition 2.8 (Field)**.** A field k is a commutative ring where every non-zero element has a multiplicative inverse.

An example of a field is a ring of integers modulo a prime. Although this holds in general, we will only show an example for one prime.

Example 6. A ring $\mathbb{Z}/p\mathbb{Z}$ under addition and multiplication, where p is a prime, is a field. First, note that $p\mathbb{Z}$ is a proper ideal by Example 3, so that $\mathbb{Z}/p\mathbb{Z}$ is indeed a ring. Now take, for example, $p = 13$. Then, the elements in this ring are all integers from 0 until 12. We can check that all nonzero integers have a multiplicative inverse: Thus each nonzero element is invertible and therefore $\mathbb{Z}/13\mathbb{Z}$ is a field.

$$\begin{array}{lcl@{\qquad}lcl} 1\cdot 1 &=& 1 & 7\cdot 2 &=& 1 \\ 2\cdot 7 &=& 1 & 8\cdot 5 &=& 1 \\ 3\cdot 9 &=& 1 & 9\cdot 3 &=& 1 \\ 4\cdot 10 &=& 1 & 10\cdot 4 &=& 1 \\ 5\cdot 8 &=& 1 & 11\cdot 6 &=& 1 \\ 6\cdot 11 &=& 1 & 12\cdot 12 &=& 1 \end{array}$$

In number theory, an important concept within fields are the roots of unity.

Definition 2.9 (nth root of unity)**.** An nth root of unity of a field $k = \mathbb{Z}/p\mathbb{Z}$ is an integer $\xi \in \mathbb{Z}/p\mathbb{Z}$ that satisfies the equation

$$\xi^n = 1.$$

Furthermore, a root of unity is said to be primitive if

$$\xi^n = 1 \text{ and } \xi^m \neq 1 \ \ \forall m \in \{1, 2, \dots n-1\}.$$

Once a primitive root ξ has been found, it follows that

$$\xi^i \text{ for } \gcd(i, n) = 1$$

are also primitive roots. Furthermore, note that the zeros of the polynomial

$$f(z) = z^n - 1$$

over a certain field k are exactly the roots of unity of this field. Extending this idea to a polynomial where each zero is a primitive root of unity yields the following definition:

Definition 2.10 (Cyclotomic polynomial)**.** The nth cyclotomic polynomial $\Phi_n(z)$ is defined by the fact that each zero is a primitive nth root of unity, with multiplicity one.

Utilizing the above definition, we can give a formula for the nth cyclotomic polynomial. Let ξ be a primitive nth root of unity over a field k, then the nth cyclotomic polynomial is given by:

$$\Phi_n(z) = \prod_{\substack{0 \leq i < n \\ \gcd(i,n)=1}} (z - \xi^i). \tag{1}$$

Note that, from this formula, it can be concluded that $\Phi_n(z)$ will be a polynomial of degree $\phi(n)$, because there are exactly $\phi(n)$ integers i less than and coprime to n. Here, $\phi(n)$ denotes the Euler totient function.

Example 7. We will give an example of the 6th cyclotomic polynomial. Using Equation 1, we can verify that

$$\Phi_6(z) = \prod_{j=1}^{2}(z - \xi_j) = (z - \xi_1)(z - \xi_2) = z^2 - (\xi_1 + \xi_2)z + \xi_1\xi_2$$

where ξ_1 and ξ_2 solve $z^6 = 1$. Then clearly, since 1 and 5 are the only numbers less than and coprime to 6, we find that $\xi_2 = \xi_1^5$. Thus,

$$\xi_1\xi_2 = \xi_1^6 = 1$$

and

$$\xi_1 + \xi_2 = 1,$$

hence we find

$$\Phi_6(z) = z^2 - z + 1.$$

2.5 Lattices

Lattices are discretizations of vector spaces. A lattice $L \subseteq \mathbb{R}^n$ is defined as:

Definition 2.11. A lattice is the set of all integer linear combinations of vectors in a basis of $\mathbb{R}^n$. For a basis $\{b_1, b_2, \ldots, b_n\}$ and coefficients $a_i \in \mathbb{Z}$, for $i \in \{1, 2, \ldots, n\}$, a lattice is defined as

$$L = \left\{ \sum_{i=1}^{n} a_i \cdot \underline{b}_i \;\middle|\; a_i \in \mathbb{Z} \right\}.$$

An important computational lattice problem is the Shortest Vector Problem (SVP), which aims to find one of the shortest non-zero vectors in a lattice. Note that this shortest vector is not unique, which becomes evident from Figure 3. While the vector in red is a shortest vector, the same vector from $(0, 0)$ pointing in the opposite direction is the same length, thus yielding multiple shortest vectors. In the case of more dimensions, there can consequently be more shortest vectors with equal length.

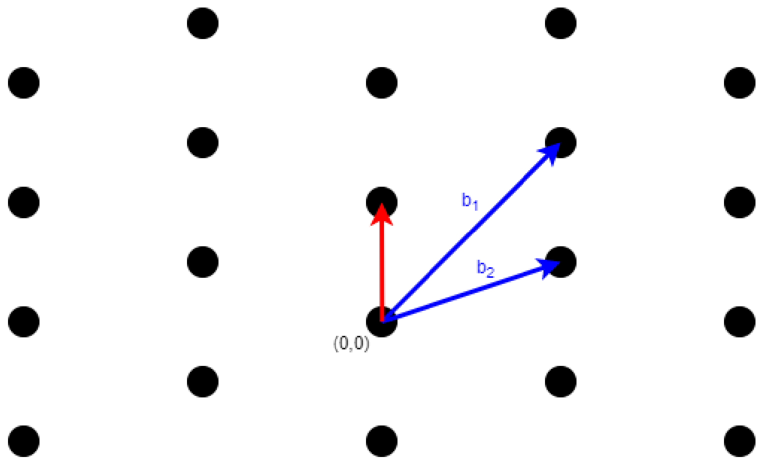

Figure 3: A lattice with basis $\{b_1, b_2\}$ and a corresponding shortest vector in red.

These SVP problems are NP-hard, but if we simply task ourselves with finding a "short enough" vector in the lattice, then there exists an algorithm that solves the problem in polynomial time. This algorithm, called the LLL basis reduction algorithm, was invented in 1982 by Lenstra, Lenstra and Lovász [26].

2.5.1 The LLL algorithm

The LLL algorithm by Lenstra, Lenstra and Lovász [26] takes a basis $B = \{b_1, b_2, \ldots, b_d\}$ for a lattice in $\mathbb{R}^n$ as input and outputs a short, near orthogonal basis of this lattice. Here, short denotes that the basis vectors have a small Euclidean norm. It first orthogonalizes the basis B using the Gram-Schmidt process. This algorithm was first introduced in 1907 by Schmidt [32], and essentially followed the same steps as an earlier version written in 1883 by Gram [15]. After this orthogonalization step, which yields a new basis $B^* = \{b_1^*, b_2^*, \ldots, b_d^*\}$, the so called Gram-Schmidt coefficients are calculated:

$$\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \quad \text{for } 1 \leq j \leq i \leq n.$$

where $\langle \cdot, \cdot \rangle$ defines the standard inner product on $\mathbb{R}^n$. Note that these Gram-Schmidt coefficients are typically not integers, and therefore they will be rounded to make sure that the new basis is still in the lattice. Now, the idea of the algorithm is to change the given basis vectors in B to shorter vectors. The algorithm has a "working vector", which is the vector that we are working to reduce. Let us call this vector b_k. Now, we can first shorten this vector by subtracting integer multiples of the vectors $b_1, \ldots, b_{k-1}$, according to the Gram-Schmidt coefficients. We call this updated vector b_k', and now we only consider b_{k-1} and b_k'. If we want to shorten b_{k-1}, we will have to subtract multiples of b_k'. But then b_{k-1} is our working vector, and thus b_k' and b_{k-1} have to be swapped. However, when we swap these two vectors, it means that we will also have to shorten it with all prior vectors again. Thus, we want to know when we should swap these vectors. We determine this using the Lovasz condition:

$$\|b_k^*\|^2 \geq \left(\delta - \mu_{k,k-1}^2\right) \|b_{k-1}^*\|^2 \tag{2}$$

where $\delta \in (\frac{1}{4}, 1)$, and is often chosen as $\frac{3}{4}$. If b_k^* satisfies this condition, then we can continue and take b_{k+1} as a working vector. If it does not, we swap b_k' and b_{k-1}. Simply stated, we take a working vector and keep shortening it until it is close enough to the previous vectors, according to the Lovasz condition. Once this condition is met, we continue with a new working vector.

The LLL algorithm has a polynomial runtime, but will not necessarily output a basis consisting of the shortest linearly independent vectors. To find better approximations of a shortest vector, other algorithms for lattice reduction exist, but these algorithms run in exponential time. Later on, we will see another lattice reduction algorithm used in an attack on NTRU+.

We will now illustrate the LLL algorithm at the hand of an example.

Example 8. Let us consider an example of a lattice L in $\mathbb{R}^2$ with basis vectors $b_1 = \begin{pmatrix} 201 \\ 37 \end{pmatrix}$ and $b_2 = \begin{pmatrix} 1648 \\ 297 \end{pmatrix}$.
Now we use Gram-Schmidt to find an orthogonal basis: $b_1^* = \begin{pmatrix} 201 \\ 37 \end{pmatrix}$ and

$$b_2^* = \begin{pmatrix} 1648 \\ 297 \end{pmatrix} - \frac{\begin{pmatrix} 1648 \\ 297 \end{pmatrix} \cdot \begin{pmatrix} 201 \\ 37 \end{pmatrix}}{\begin{pmatrix} 201 \\ 37 \end{pmatrix} \cdot \begin{pmatrix} 201 \\ 37 \end{pmatrix}} \cdot \begin{pmatrix} 201 \\ 37 \end{pmatrix} = \begin{pmatrix} \frac{47323}{41770} \\ \frac{-257079}{41770} \end{pmatrix} \approx \begin{pmatrix} 1.13 \\ -6.15 \end{pmatrix}.$$

Note that this orthogonal basis vector b_2^* is not in L. Now, we will shorten our working vector b_2 using b_1. For this, we have to make sure that the shortened vector is still in the lattice, and thus we use the rounded product, which rounds any real number to the closest integer:

$$\left\lfloor \frac{\begin{pmatrix} 1648 \\ 297 \end{pmatrix} \cdot \begin{pmatrix} 201 \\ 37 \end{pmatrix}}{\begin{pmatrix} 201 \\ 37 \end{pmatrix} \cdot \begin{pmatrix} 201 \\ 37 \end{pmatrix}} \right\rceil = 8.$$

so then we update

$$b_2' = \begin{pmatrix} 1648 \\ 297 \end{pmatrix} - 8 \cdot \begin{pmatrix} 201 \\ 37 \end{pmatrix} = \begin{pmatrix} 40 \\ 1 \end{pmatrix}$$

Now, we check the Lovász condition, from Equation 2, with $\delta = \frac{3}{4}$:

$$\|b_1^*\|^2 = 41770, \qquad \|b_2^*\|^2 = 39.16,$$
$$\mu_{2,1} = \frac{\begin{pmatrix} 40 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 201 \\ 37 \end{pmatrix}}{\|b_1^*\|^2} = \frac{8077}{41770} \text{and thus}$$
$$39.16 \ngeq \left(\frac{3}{4} - \frac{8077}{41770}^2 \right) \cdot 41770 \approx 29765.7.$$

So we swap the vectors b_1 and b_2, such that

$$b_1 = \begin{pmatrix} 40 \\ 1 \end{pmatrix} \qquad b_2 = \begin{pmatrix} 201 \\ 37 \end{pmatrix}$$

and we once again use Gram-Schmidt to find the corresponding orthogonalized basis vectors.

$$b_1^* = \begin{pmatrix} 40 \\ 1 \end{pmatrix} \qquad b_2^* = \begin{pmatrix} 201 \\ 37 \end{pmatrix} - \frac{\begin{pmatrix} 201 \\ 37 \end{pmatrix} \cdot \begin{pmatrix} 40 \\ 1 \end{pmatrix}}{\begin{pmatrix} 40 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 40 \\ 1 \end{pmatrix}} \cdot \begin{pmatrix} 40 \\ 1 \end{pmatrix} \approx \begin{pmatrix} -0.80 \\ 32.0 \end{pmatrix}$$

and we shorten b_2 as

$$b_2 = \begin{pmatrix} 201 \\ 37 \end{pmatrix} - \left\lfloor \frac{\begin{pmatrix} 201 \\ 37 \end{pmatrix} \cdot \begin{pmatrix} 40 \\ 1 \end{pmatrix}}{\begin{pmatrix} 40 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 40 \\ 1 \end{pmatrix}} \right\rceil \cdot \begin{pmatrix} 40 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 32 \end{pmatrix}$$

Checking the Lovász condition tells us that we are not yet done, and we should once again swap such that

$$b_1 = \begin{pmatrix} 1 \\ 32 \end{pmatrix} \qquad b_2 = \begin{pmatrix} 40 \\ 1 \end{pmatrix}$$

with Gram-Schmidt basis:

$$b_1^* = \begin{pmatrix} 1 \\ 32 \end{pmatrix} \qquad b_2^* \approx \begin{pmatrix} 39.9 \\ -1.3 \end{pmatrix}$$

Shortening b_2 now yields b_2 again, i.e., the vector is shortened as far as possible, since

$$\left\lfloor \frac{\begin{pmatrix} 40 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 32 \end{pmatrix}}{\begin{pmatrix} 1 \\ 32 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 32 \end{pmatrix}} \right\rceil = 0$$

Finally, checking the Lovász condition gives:

$$\|b_1^*\|^2 = 1025 \qquad \|b_2^*\|^2 \approx 1593.7$$
$$\mu_{2,1} = \frac{\begin{pmatrix} 40 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 32 \end{pmatrix}}{\|b_1^*\|^2} \approx 0.070$$

and hence

$$\|b_2^*\|^2 \geq \left(\frac{3}{4} - \mu_{2,1}^2\right) \|b_1^*\|^2$$
$$1593.7 \geq \left(\frac{3}{4} - 0.070^2\right) \cdot 1025$$

Now, we can move onto our next basis vector. But since this was a two dimensional example, the algorithm terminates, indicating that we have found a shortened basis

$$\left\{ \begin{pmatrix} 40 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 32 \end{pmatrix} \right\}.$$

3 NTRU variants

Over time, many different variants of NTRU have been developed. In this thesis, we take a look at the original NTRUEncrypt, better known as just NTRU. Then, we will examine the improved NTRU Prime, which was introduced for the NIST competition, the NTTRU scheme which incorporates Number Theoretic Transform for faster arithmetic, and finally the new NTRU+ that was submitted to the Korean Post Quantum Competition. Note that, in this chapter, some of these variants are presented as PKEs, while others are presented as KEMs. The reason for this is that the PKE systems for NTRU and NTTRU show the necessary mechanics, while NTRU Prime was only presented as a KEM when it was submitted to the NIST competition. NTRU+ is presented as both a PKE and a KEM by its authors, however we only discuss the KEM version in this thesis, because it contains the most possibilities for research.

3.1 NTRU

NTRU is a lattice-based PKE algorithm proposed by Hoffstein, Pipher and Silverman in 1998 [18].

3.1.1 Parameters

NTRU is defined using a ring

$$R = \mathbb{Z}[x]/(x^n - 1)$$

where n is often, but not necessarily, prime. NTRU also works with rings

$$R_3 = (\mathbb{Z}/3\mathbb{Z})[x]/(x^n - 1)$$
$$R_q = (\mathbb{Z}/q\mathbb{Z})[x]/(x^n - 1)$$

where q is usually a power of two, but in any case $\gcd(q, 3) = 1$. Finally, a $t \in \mathbb{N}$ is chosen, which will dictate the number of nonzero coefficients for each polynomials during the key generation.

3.1.2 Key generation

For key generation, one takes two polynomials f and g in R with coefficients in $\{-1, 0, 1\}$. The polynomial f has t coefficients equal to 1 and $t-1$ coefficients equal to -1, while g has t coefficients equal to 1 and also t equal to -1. The rest of the coefficients in both polynomials are 0. The public key h is then defined as

$$h = 3gf^{-1} \in R_q$$

where f^{-1} is the inverse of f in R_q. Note that if this inverse does not exist, a new f should be generated while the previous one is discarded. The private key then consists of the pair f and f_3, which is the inverse of f in R_3 such that $f \cdot f_3 = 1 \mod 3$. Again, if this inverse does not exist, f is discarded and a new polynomial has to be chosen. If inverting f works in both rings, the keypair then consists of h as public key and the pair (f, f_3) as private key.

3.1.3 Encryption

Encryption takes the public key and a message $m \in R$ with coefficients in $\{-1, 0, 1\}$. The algorithm creates a ciphertext c based on this m and using a randomly generated polynomial r with coefficients in $\{-1, 0, 1\}$, as

$$c = r \cdot h + m \in R_q.$$

Here, r is often chosen under the same conditions as g, so t coefficients as 1 and t equal to -1.

3.1.4 Decryption

Decryption of the ciphertext c works as follows: we calculate $a \in R_q$ as

$$a = f \cdot c \mod {}^{\pm}q$$

where we let mod ${}^{\pm}q$ denote that each coefficient of a mod q is moved to $\left(-\frac{q}{2}, \frac{q}{2}\right]$. Then, we find m again by calculating

$$m = f_3 \cdot a \in R_3$$

where we once again move the coefficients to $\{-1, 0, 1\}$ by applying mod ${}^{\pm}3$.

But note that combining different modular reductions generally will cause problems. After all,

$$11 \mod 3 \mod 2 = 0$$

while

$$11 \mod 2 \mod 3 = 1.$$

Therefore, a restriction has to be made. We require a, which we write out using $c = r \cdot h + m$, so

$$a = 3 \cdot r \cdot g + f \cdot m \in R, \tag{3}$$

to only have coefficients $a_i \in \left(\frac{q}{2}, \frac{q}{2}\right]$. Only then, no modulo reduction is needed. Note that r, g, f and m are all polynomials with coefficients in $\{-1, 0, 1\}$, so this requires choosing $\frac{q}{2}$ larger than the absolute maximum coefficient a can have. So if r, g and f all contain d non-zero coefficients, then the maximum absolute value of a coefficient of $3 \cdot r \cdot g + f \cdot m$ will be

$$3d + d,$$

so we will have to pick q such that

$$\frac{q}{2} > 4d.$$

Note that, since $d = 2 \cdot t$ for r and g while $d = 2t - 1$ for f, this means we require a q such that

$$q > 16t.$$

If q is not chosen in this way, then it is possible that a reduction modulo q occurs for the calculation of a. In that case, the decrypted message will be different from the encrypted message, as the $\mod {}^{\pm}3$ still follows after the $\mod q$.

3.1.5 Example

To further illustrate how the NTRU key generation, encryption and decryption works, we give an example with small parameters:

Let us take $n = 7$ and $q = 32$. This gives the rings

$$R = \mathbb{Z}[x]/(x^7 - 1)$$
$$R_3 = (\mathbb{Z}/3\mathbb{Z})[x]/(x^7 - 1)$$
$$R_{32} = (\mathbb{Z}/32\mathbb{Z})[x]/(x^7 - 1)$$

Take $t = 3$ and let us use this to generate the random polynomials:

$$f = x^6 + x^5 - x^3 + x - 1$$
$$g = -x^6 + x^5 - x^4 - x^3 + x^2 + 1$$

where f is invertible in R_3 and R_{32}. First, calculate the inverse of f in R_3:

$$f_3 = x^5 + x^4 + 2x^3 + 2x^2 + 2x + 2$$

Now, we calculate the inverse of f in R_{32}:

$$f^{-1} = -2x^6 - 4x^5 + 9x^4 - 16x^3 - 4x^2 + x - 15.$$

Then

$$\begin{aligned}
h &= \frac{3g}{f} = 3 \cdot g \cdot f^{-1} \\
&= 3 \cdot (-x^6 + x^5 - x^4 - x^3 + x^2 + 1) \cdot (-2x^6 - 4x^5 + 9x^4 - 16x^3 - 4x^2 + x - 15) \\
&= 6x^{12} + 6x^{11} - 33x^{10} + 93x^9 - 57x^8 - 6x^7 + 129x^6 - 96x^5 + 57x^4 - 57x^2 + 3x - 45 \\
&= x^6 + 6x^5 + 31x^4 + 31x^3 + 4x^2 + 10x + 13 \in R_{32}.
\end{aligned}$$

With that, key generation is complete and we can start to encrypt a message m. Let us pick a random message $m \in R_3$ with

$$m = x^4 - x^3 + x - 1.$$

We can encrypt this message as

$$c = r \cdot h + m \in R_{32},$$

which also requires a random small polynomial r. We take

$$r = -x^6 + x^5 - x^4 + x^3 + x^2 - x.$$

Then the ciphertext is

$$\begin{aligned}
c &= r \cdot h + m \in R_{32} \\
&= (-x^6 + x^5 - x^4 + x^3 + x^2 - x) \cdot (x^6 + 6x^5 + 31x^4 + 31x^3 + 4x^2 + 10x + 13) + x^4 - x^3 + x - 1 \\
&= -x^{12} - 5x^{11} - 26x^{10} - 5x^9 + 3x^8 - x^7 + 49x^6 + 7x^5 - 30x^4 + 19x^3 + 3x^2 - 13x + x^4 - x^3 + x - 1 \\
&= 17x^6 + 6x^5 + 30x^4 + 24x^3 + 30x^2 + 23x + 30.
\end{aligned}$$

Finally, we can decrypt this message again. By computing

$$\begin{aligned}
a &= f \cdot c \in R_{32} \\
&= 17x^{12} + 23x^{11} + 36x^{10} + 37x^9 + 48x^8 + 40x^7 + 18x^6 + 24x^5 - 29x^4 - 24x^3 - 7x^2 + 7x - 30 \\
&= 18x^6 + 9x^5 + 26x^4 + 12x^3 + 30x^2 + 23x + 10
\end{aligned}$$

and then we take all coefficients $\mod {}^{\pm}q$, which yields

$$a = -14x^6 + 9x^5 - 6x^4 + 12x^3 - 2x^2 - 9x + 10 \mod {}^{\pm}q.$$

Finally, as stated above, the inverse of f in R_3 is

$$f_3 = x^5 + x^4 + 2x^3 + 2x^2 + 2x + 2.$$

This allows us to calculate

$$\begin{aligned}
m &= f_3 \cdot a \in R_3 \\
&= (x^5 + x^4 + 2x^3 + 2x^2 + 2x + 2) \cdot (-14x^6 + 9x^5 - 6x^4 + 12x^3 - 2x^2 - 9x + 10) \\
&= -14x^{11} - 5x^{10} - 25x^9 - 4x^8 - 12x^7 - 9x^6 + 27x^5 + 22x^3 - 2x^2 + 2x + 20 \\
&= -9x^6 + 27x^5 - 14x^4 + 17x^3 - 27x^2 - 2x + 8 \\
&= x^4 - x^3 + x - 1 \mod {}^{\pm}3,
\end{aligned}$$

which is indeed our original m. In this example, it should be noted that $q = 32$ is too small to guarantee no decryption failures for $t = 3$. However, because the possibility that the maximum coefficient obtained from Equation 3 equals $6t+2t$ is a worst-case scenario, where the coefficients equal to 1 meet other positive coefficients, while -1 meets -1, this choice of parameters will work for most polynomials.

3.1.6 Connection to lattices

Even though it might not be immediately clear from the description, NTRU is a lattice-based cryptographic system. Lattice structures have previously been discussed in Section 2.11. As discussed in Chapter 1, the interest in lattice-based cryptography results from the danger of Shor's algorithm and quantum computers successfully defeating many current schemes such as RSA and Diffie-Hellman. Instead, lattice-based systems rely on the Shortest Vector Problem (SVP) or the Closest Vector Problem (CVP) that are assumed to be difficult to solve even with a quantum computer. For NTRU, the lattice used has basis

$$\begin{bmatrix} qI_n & 0 \\ H & I_n \end{bmatrix}$$

where H corresponds to multiplying with $\frac{h}{3}$ and I_n is the $n \times n$ identity matrix, as explained, e.g., by Nitaj in 2015 [30]. Now, the pair (f, g) is a short vector in this lattice. To see this, note that $h = \frac{3g}{f}$ in R_q means $\frac{f \cdot h}{3} \equiv g \bmod q$, i.e., there exists some $k \in R$ with $\frac{f \cdot h}{3} = g - kq$ in R. For this k, it then holds that

$$\begin{bmatrix} k & f \end{bmatrix} \cdot \begin{bmatrix} qI_n & 0 \\ H & I_n \end{bmatrix} = \begin{bmatrix} k \cdot q + f \cdot H & f \end{bmatrix} = \begin{bmatrix} g & f \end{bmatrix}.$$

Similarly there exists some $k' \in R$ so that encryption corresponds to

$$\begin{bmatrix} k' & 3r \end{bmatrix} \cdot \begin{bmatrix} qI_n & 0 \\ H & I_n \end{bmatrix} + \begin{bmatrix} m & -3r \end{bmatrix} = \begin{bmatrix} k'q + hr + m & 0 \end{bmatrix} = \begin{bmatrix} c & 0 \end{bmatrix}$$

which is a vector close to the lattice vector $\begin{bmatrix} k'q + hr & 3r \end{bmatrix}$. Hence, decryption amounts to solving the CVP for $\begin{bmatrix} c & 0. \end{bmatrix}$.

3.2 NTRU Prime

NTRU Prime is an NTRU variant that attempts to limit the attack surface, introduced by Bernstein, Chuengsatiansup, Lange and van Vredendaal in 2017 [5]. It was a submission to the NIST Post Quantum Competition where it survived until round 3. In this section, we will examine the Streamlined NTRU Prime system as it was submitted for the third round. Streamlined NTRU Prime, will henceforth be referred to as just NTRU Prime. NTRU Prime is a KEM. Therefore, there is no message m being encrypted, and instead the aim is to share a secret, small polynomial r, which can then be used to obtain a shared symmetric key.

3.2.1 Parameters

NTRU Prime starts by defining a parameter $w \in \mathbb{Z}$, which will define the number of nonzero coefficients in f and r. Such a polynomial, with w nonzero coefficients that are either -1 or 1, is called small. Moreover, NTRU Prime requires a prime q with $q \geq 16w + 1$. This condition ensures that no decryption failures can occur. Then, a prime n, chosen such that $2n \geq 3q$, defines the polynomial

$$x^n - x - 1,$$

which is irreducible modulo q. With this irreducible polynomial, the field

$$R_q = (\mathbb{Z}/q\mathbb{Z})[x]/(x^n - x - 1)$$

is defined. In the encapsulation and decapsulation process, we will also need the rings

$$R = \mathbb{Z}[x]/(x^n - x - 1)$$

and

$$R_3 = (\mathbb{Z}/3\mathbb{Z})[x]/(x^n - x - 1).$$

3.2.2 Key generation

The private key is f, which is taken as a small polynomial in R with w nonzero coefficients, which can either be 1 or -1. NTRU Prime also defines a polynomial g in R with coefficients in $\{-1, 0, 1\}$, and such that g is invertible in R_3 and R_q. Observe that any nonzero polynomial is invertible in R_q , since it is a field.

The public key is then

$$h = \frac{g}{3f} \in R_q.$$

The private key is again a pair of polynomials, this time f in R and g^{-1} in R_3.

3.2.3 Encapsulation

Encapsulation works using a so-called rounded ciphertext: a small polynomial with w nonzero coefficients, which is called r, is generated and used to compute

$$h \cdot r \in R_q$$

The ciphertext is then this polynomial, with each coefficients viewed in $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$, and thereafter rounded to the nearest multiple of 3, which we will call c. This can be seen as the same encryption method used in NTRU,

$$c = r \cdot h + m \in R_q \tag{4}$$

but here the m in Equation 4 is chosen such that each coefficient of c is in a restricted subset of $\mathbb{Z}/q\mathbb{Z}$.

3.2.4 Decapsulation

Decapsulation takes the ciphertext c and the private key f, alongside the polynomial $v = g^{-1} \in R_3$, and first computes

$$3f \cdot c = 3f \cdot m + g \cdot r \in R_q.$$

Now, we convert each coefficient of this polynomial to an integer in $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$ and reduce modulo 3, which yields a polynomial $e \in R_3$. Finally, we compute

$$r' = e \cdot v \in R_3$$

and output r' if r' has weight w. This ensures that the ciphertext has not been tampered with, while the weight w protects against decryption failures by ensuring that a reduction modulo q is not necessary when calculating $3f \cdot m + g \cdot r$.

3.3 NTTRU

In 2019, Lyubashevsky and Seiler proposed an NTRU variant that uses Number Theoretic Transforms (NTT) for computations, which they conveniently called NTTRU [27].

3.3.1 Parameters

NTTRU uses a ring that allows for the use of NTTs, which is of the form

$$R = \mathbb{Z}[x]/(x^n - x^{n/2} + 1)$$

where n is of the form $2^i \cdot 3^j$, with $i, j > 0$. Here, $x^n - x^{n/2} + 1$ is a cylcotomic trinomial. We have already seen an example of such a trinomial in Example 7. NTTRU moreover uses the rings

$$R_3 = (\mathbb{Z}/3\mathbb{Z})[x]/(x^n - x^{n/2} + 1)$$
$$R_q = (\mathbb{Z}/q\mathbb{Z})[x]/(x^n - x^{n/2} + 1)$$

where q is a prime which satisfies that $q - 1$ is divisible by n.

3.3.2 Number Theoretic Transform

In the NTTRU KEM system, all polynomials are stored in the Number Theoretic Transform representation. This representation originates from the fact that $x^n - x^{n/2} + 1$ is reducible over $\mathbb{Z}/q\mathbb{Z}$. We split

$$x^n - x^{n/2} + 1 = (x^{n/2} - \xi_1)(x^{n/2} - \xi_2)$$

where ξ_1 and $\xi_2 = \xi_1^5$ are the two primitive sixth roots of unity of the underlying ring. For this, recall Example 7.

Now, we can continue to split these factors down to smaller order polynomials. Note that, since our $n = 2^i \cdot 3^j$, can let $m = 2^{i-1} \cdot 3^j$ such that any

$$x^m - \xi_1$$

can be split as

$$x^m - \xi_1 = (x^{m/2} + \sqrt{\xi_1})(x^{m/2} - \sqrt{\xi_1}) \mod q$$

or

$$x^m - \xi_1 = (x^{m/3} - \sqrt[3]{\xi_1})(x^{m/3} - \omega\sqrt[3]{\xi_1})(x^{m/3} - \omega^2\sqrt[3]{\xi_1}) \mod q,$$

depending on i and j. Note that $\sqrt{\xi_1}$ are twelfth roots of unity of the ring R_q, while $\sqrt[3]{\xi_1}$ are the eighteenth and ω the third primitive roots of unity. Moreover, these splits of the polynomial form a ring isomorphism

$$(\mathbb{Z}/q\mathbb{Z})[x]/(x^n - x^{n/2} + 1) \cong (\mathbb{Z}/q\mathbb{Z})[x]/(x^{n/2} - \xi_1) \times (\mathbb{Z}/q\mathbb{Z})[x]/(x^{n/2} - \xi_2)$$

by the Chinese Remainder Theorem (CRT) [27]. Since NTTRU uses $n = 2^i \cdot 3^j$, and n divides $q-1$, we can factor our original ring to a product of rings where each quotient polynomial has degree at most 3. For an explanation see below. For NTTRU, the authors suggest using a prime $q = 7681$, where the Euler totient is

$$\phi(q) = q - 1 = 7680 = 2^9 \cdot 3 \cdot 5.$$

Because q is prime, this makes $\mathbb{Z}/q\mathbb{Z}$ a field, as shown in Example 6, and therefore it has a multiplicative cyclic group with a generator g. Now, if n divides $q - 1$ then

$$g^{\frac{q-1}{n}}$$

is a primitive nth root of unity in $\mathbb{Z}/q\mathbb{Z}$. So to guarantee we can split the cyclotomic trinomial into polynomials of degree 3, we must have that n divides q. If instead we demanded linear factors, we must thus have that $3n$ divides q. The proposed n for NTTRU is 768, which factors as $2^8 \cdot 3$, thus n divides $\phi(q)$. Therefore, there are sufficiently many primitive roots of unity in the field, and the cyclotomic trinomial factors into factors of at most degree 3.

Using this isomorphism, which factors the ring into rings with a quotient of degree at most 3, NTTRU represents polynomials in R_q as polynomials in the isomorphic ring by using Montgomery multiplication. This algorithm for fast modular multiplication without trial division, was introduced by Montgomery in 1985 [29]. Montgomery representations in modular arithmetic with modulus q take a number B that is larger than and coprime to q. Then, the Montgomery form represents the class of $a \mod q$ as $aB \mod q$. Now, addition and subtraction work the same as ordinary modular addition and subtraction:

$$\begin{aligned} aB + bB &\equiv (a+b)B \mod q \\ aB - bB &\equiv (a-b)B \mod q \end{aligned}$$

However, multiplication in Montgomery representation works slightly differently. First, take B^{-1} as the multiplicative inverse of $B \mod q$. Then multiplication such that $c \equiv a \cdot b \mod q$ is defined in Montgomery Representation as

$$cB \equiv (aB \cdot bB)\, B^{-1} \equiv abB \mod q.$$

With the Montgomery multiplication and background theory on roots of unity, the NTT can now be constructed. NTTRU uses a 2018 paper by Seiler [35], where the NTT and inverse NTT are described in Algorithm 1 and 2.

3.3.3 Key generation

NTTRU generates polynomials f' and g using the Centered Binomial Distribution (CBD), which is defined as follows:

Definition 3.1 (Centered Binomial Distribution)**.** The CBD takes a number of bits as input and outputs an integer. We assign bits b_i and b'_i randomly from $\{0,1\}$, for $i \in \{1,2,\dots,k\}$. Then the CBD calculates and returns

$$\psi_k = \sum_{i=1}^{k} b_i - b'_i.$$

The coefficients of the polynomials f' and g are generated following ψ_2, which generates the random bits b_1, b_2, b'_1, b'_2 and then computes

$$b_1 + b_2 - b'_1 - b'_2 \mod {}^{\pm}3$$

for each coefficient in f'. Here $\mod {}^{\pm}3$ denotes the balancing of this sum around 0, so only taking values in $\{-1, 0, 1\}$. In practice, this CBD will take the values:

$$\psi_2 = \begin{cases} -1 & \text{w.p. } \frac{5}{16} \\ 0 & \text{w.p. } \frac{3}{8} \\ 1 & \text{w.p. } \frac{5}{16} \end{cases}$$

With f', the polynomial f is computed as

$$f = 3 \cdot f' + 1.$$

Then, as we have seen previously,

$$h = \frac{3g}{f} \in R_q$$

where h will serve as the public key and f as the private key.

3.3.4 Encryption

For encryption, NTTRU generates a random polynomial r with coefficients in $\{-1, 0, 1\}$ according to ψ_2. Furthermore, a message m is generated, once again according to the distribution ψ_2. Then, the ciphertext is computed as

$$c = r \cdot h + m \in R_q.$$

3.3.5 Decryption

Decryption in NTTRU works by calculating

$$\begin{aligned} a &= f \cdot c \in R_q \\ &= 3 \cdot g \cdot r + f \cdot m \mod {}^{\pm}q \\ &= 3 \cdot g \cdot r + 3f' \cdot m + m \mod {}^{\pm}q \end{aligned}$$

and then

$$m = a \mod {}^{\pm}3$$

where the $^{\pm}$ once again denotes a balanced modulo operation. Note that there is no need for a multiplication by the inverse of f in R_3, as f is defined as $3f' + 1$. This idea originates from a paper by Hoffstein and Silverman from 2001 [20]. We will discuss decryption failures in the next section but remark that using coefficients generated by the CBD means that there is no upper bound other than n on the number of non-zero coefficients.

3.4 NTRU+

NTRU+, proposed by Kim and Park in 2022 [23], is a new cryptographic system that was sent in as an entry to the Korean Post Quantum Cryptography Competition. In their paper, Kim and Park propose both a PKE NTRU+ system and an NTRU+ KEM, where they claim to achieve CCA security. We will compare the NTRU+ PKE to the aforementioned PKE systems, but we will also introduce the NTRU+ KEM as this contains some newly introduced transforms. Therefore, Sections 3.4.4 and 3.4.5 will cover NTRU+ as a PKE system, while Sections 3.4.6 and 3.4.7 contain NTRU+ as a CCA secure KEM.

3.4.1 Parameters

NTRU+ uses a ring that allows for computations using Number Theoretic Transforms (NTT). Here, we again need a restriction on q such that there exist enough primitive roots of unity, and we can split the cyclotomic trinomial into small enough polynomials.

This idea is taken directly from the NTTRU system, which is described above, and again uses the ring

$$R = \mathbb{Z}[x]/(x^n - x^{n/2} + 1)$$

where $x^n - x^{n/2} + 1$ is cyclotomic trinomial, i.e., a unique irreducible polynomial over $\mathbb{Z}$. Here, the integer $n = 2^i \cdot 3^j$ for $i, j \in \mathbb{N}$, and with $i, j > 0$ so n is always even. It furthermore relies on the rings

$$\begin{aligned} R_3 &= (\mathbb{Z}/3\mathbb{Z})[x]/(x^n - x^{n/2} + 1) \\ R_q &= (\mathbb{Z}/q\mathbb{Z})[x]/(x^n - x^{n/2} + 1) \end{aligned}$$

where q is not necessarily prime, but it is coprime to 3.

Now, we investigate the requirements that the Number Theoretic Transform demands from q. Similar to NTTRU, the NTRU+ authors take a prime $q = 3457$. Then, the Euler totient is

$$\phi(q) = 3456 = 2^7 \cdot 3^3.$$

The NTRU+ authors propose four different options for the parameter n, all using this same q. These are

$$\begin{aligned} 576 &= 2^6 \cdot 3^2 \\ 768 &= 2^8 \cdot 3 \\ 864 &= 2^5 \cdot 3^3 \\ 1152 &= 2^7 \cdot 3^2. \end{aligned}$$

Now, since q is prime, then similar to Section 3.3.2 the ring $\mathbb{Z}/q\mathbb{Z}$ is a field with a cyclic multiplicative group. The NTRU+ authors want to factor to second or third degree polynomials. For second degree factors, they require the $\frac{3n}{2}$th roots of unity in the ring, i.e., we must have that $\frac{3n}{2}$ divides $\phi(q)$. For third degree factors, we once again require n to divide q. Thus, note that 864 can use degree 3 factors at most, while the others can be split further to factors of degree 2, and even linear for the first and last option.

3.4.2 ACWC$_2$

Kim and Park, the authors of NTRU+ , introduce a transform that takes a cryptographic system with an average chance at decryption failure to a minimal chance of decryption failure. They call this transform the Average Case to Worst Case 2 (ACWC$_2$) transform, for which the authors present a proof of its effectiveness. The details of the proof fall outside the scope of this thesis, hence we refer to Section 3.2 of the paper by Kim and Park for the specifics [23]. However, the ACWC$_2$ utilizes a function called the Semi-generalized One Time Pad (SOTP), which is of interest to us, as it presents an opportunity for further research into the system's security.

The SOTP, which is an invertible function, works as follows:

$$\begin{aligned} \mathsf{SOTP} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n &\to \{-1,0,1\}^n \\ (x, u_1, u_2) &\mapsto y = (x \oplus u_1) - u_2 \end{aligned}$$

and its inverse

$$\begin{aligned} \mathsf{Inv} : \{-1,0,1\}^n \times \{0,1\}^n \times \{0,1\}^n &\to \{0,1\}^n \\ (y, u_1, u_2) &\mapsto x = (y + u_2) \oplus u_1 \end{aligned}$$

Here, $\oplus$ is the bitwise XOR operator. Also observe that the SOTP maps from a binary space to a ternary space. We can verify that the Inv function does indeed yield us x again:

$$\begin{aligned} \mathsf{Inv}(\mathsf{SOTP}(x, u_1, u_2), u_1, u_2) &= \mathsf{Inv}((x \oplus u_1) - u_2, u_1, u_2) \\ &= ((x \oplus u_1) - u_2 + u_2) \oplus u_1 \\ &= (x \oplus u_1) \oplus u_1 \\ &= x. \end{aligned}$$

Although the SOTP and Inv functions are defined on vectors, they are extended to accept polynomial inputs by taking the coefficients as a vector. In this case, the constant coefficient is the first entry of the vector, the first degree coefficient is the second entry and so forth, until a length n vector is formed.

By applying the SOTP to the general NTRU+ instantiation, Kim and Park intend to arrive at a PKE with worst-case correctness error. To transform this PKE into an IND-CCA secure KEM, they apply the Fujisaki-Okamoto (FO) transform, which was proposed by Fujisaki and Okamoto in 1999 [14]. Recall that in Section 2.2, it is mentioned that a PKE is generally much slower than symmetric encryption, because of more complicated encryption and decryption processes. So by changing a PKE system to a KEM, the slower encryption and decryption with the PKE system only has to be executed once to obtain a shared key K. Thereafter, communication goes via a much faster symmetric encryption algorithm.

The FO transform introduces checks at the end of the decapsulation process, which verify that the decapsulation happened correctly and the ciphertext was not forged. Normally, a FO transform checks whether the $\mathsf{Enc}(pk, m) =$

c, however Kim and Park show in [23] that the use of the FO transform on NTRU+ does not require this check, and we instead recalculate the secret r' which depends on the secret message m^+. By checking that this recalculated secret r' is equal to the original r, they claim that it can be ensured that the cipher text was formed correctly. They call this variant of the Fujisaki-Okamoto transform, without re-encryption, the $\overline{\text{FO}}$.

3.4.3 Key generation

For the creation of polynomials, NTRU+, like NTTRU, uses the Centered Binomial Distribution (CBD), which is described in Definition 3.1. Coefficients of the polynomials f', g and r are chosen according to the CBD ψ_1.

Then, the CBD ψ_1 then takes the values:

$$\psi_1 = \begin{cases} -1 & \text{w.p.} \quad \frac{1}{4} \\ 0 & \text{w.p.} \quad \frac{1}{2} \\ 1 & \text{w.p.} \quad \frac{1}{4} \end{cases}$$

Now, we take the polynomial f' with coefficients in $\{-1, 0, 1\}$ according to the CBD ψ_1^n and compute the private key as

$$f = 3 \cdot f' + 1.$$

If f is not invertible in R_q, then we sample a new f' and try again. Another invertible polynomial g is also sampled according to ψ_1^n. Finally, we once again calculate the public key h as

$$h = \frac{3g}{f} \in R_q.$$

Then, the keypair consists of h as the public key and f as the secret key.

3.4.4 Encryption

Take a message $m^+ \in \{0,1\}^n$ and generate a random polynomial r according to the Centered Binomial Distribution ψ_1^n. Then, the SOTP is used, as described in Section 3.4.2, to create m as an element of $\{-1, 0, 1\}^n$ as

$$m = \text{SOTP}(m^+, G(r))$$

where G is a hash function with an output of length $2n$, which takes a bitwise representation of r as input, such that $G(r) = u_1 \| u_2$. In the C implementation, SHA256 is used for this. The ciphertext polynomial c is thereafter calculated as

$$c = r \cdot h + m \in R_q.$$

3.4.5 Decryption

The decryption aims to once again obtain the secret message m^+ from the ciphertext c. First, m is obtained by calculating

$$\begin{aligned} a &= f \cdot c \in R_q \\ &= f \cdot (h \cdot r + m) \mod {}^{\pm}q \\ &= 3 \cdot g \cdot r + 3f' \cdot m + m. \end{aligned}$$

Then,

$$m = a \mod {}^{\pm}3$$

where mod $^{\pm}q$ again centers the coefficients around 0, and $^{\pm}3$ centers the coefficients in $\{-1, 0, 1\}$ around 0. Note that this holds since $f = 3f' + 1$, thus simply taking this result in R_3 again yields m.

With this m, the secret polynomial r can be reconstructed as

$$r = (c - m) \cdot h^{-1}.$$

Finally, the Inv function of the SOTP now yields the m^+ as

$$m^+ = \mathsf{Inv}(m, G(r)).$$

Recall from Section 3.1.4 that decryption failures can happen in NTRU. Similarly, in the NTRU+ PKE, decryption failures happen if

$$a = 3 \cdot g \cdot r + 3f' \cdot m + m,$$

when computed in R, contains coefficients falling outside $\left(-\frac{q}{2}, \frac{q}{2}\right]$. Thus, to lower the chance of decryption failures happening, the authors of NTRU+ state that they implement the ACWC$_2$ transform. According to them, this transform, using the SOTP function, achieves a worst-case correctness error, meaning that decryption failures will hardly ever take place if the ciphertext is not ill-formed.

3.4.6 Encapsulation

In the KEM setting, the message m^+ is generated in R with coefficients in $\{0, 1\}$. They generate the shared symmetric key K and the secret polynomial r as the halves of the hash of m^+, so

$$(r, K) = H(m^+)$$

where H is a cryptographic hash function, such that the length of the output is $2 \cdot n$. In the C implementation, this is chosen to be SHA512. Once again, we sample m as an element of $\{-1, 0, 1\}^n$ using the SOTP and the hash of r, so

$$m = \mathsf{SOTP}(m^+, G(r))$$

where $G(r)$ is again $u_1 \| u_2$.

Thereafter, encryption works just like in NTRU, with the ciphertext c being computed by

$$c = r \cdot h + m \in R_q.$$

Then, c will be sent while K is kept as the shared symmetric key.

3.4.7 Decapsulation

For the decapsulation of the symmetric key, NTRU+ needs to recover the original m^+, but for that we first need m, which is

$$m = f \cdot c \mod {}^{\pm}q \mod {}^{\pm}3$$

where mod $^{\pm}q$ or $^{\pm}3$ again centers the coefficients around q or 3, respectively. Note that obtaining m works identically to Section 3.4.5.

Then, r is obtained as

$$r = (c - m) \cdot h^{-1}$$

Now, the Inv function, related to the SOTP explained in Section 3.4.2, yields the message m^+ by computing

$$m^+ = \mathsf{Inv}(m, G(r)).$$

With m^+ recovered, the decrypter can find K and r', as

$$(r', K) = H(m^+).$$

Finally, they check whether $r' = r$, and if so then K is correct and can be used for symmetric and asymmetric decryption.

3.4.8 Example

To illustrate how the above system works in practice, we will give an example of the NTRU+ KEM with small parameters.

Example 9 (NTRU+)**.** We start by choosing our parameters. Let $n = 2 \cdot 3 = 6$ and $q = 19$. This results in the rings

$$\begin{aligned} R &= \mathbb{Z}[x]/(x^6 - x^3 + 1) \\ R_3 &= (\mathbb{Z}/3\mathbb{Z})[x]/(x^6 - x^3 + 1) \\ R_{19} &= (\mathbb{Z}/19\mathbb{Z})[x]/(x^6 - x^3 + 1). \end{aligned}$$

Let us generate an f' according to the Centered Binomial Distribution, which yields

$$f' = -x^3 - x$$

such that

$$f = 3f' + 1 = -3x^3 - 3x + 1$$

and f is invertible in R_{19}. Furthermore, we generate a polynomial g, which is also invertible in R_{19}:

$$g = x^5 - x^2 + 1.$$

We find f^{-1} in R_{19}, yielding

$$f^{-1} = 11x^5 + 13x^4 + 4x^3 + 5x^2 + 13$$

and use it to calculate

$$\begin{aligned} h &= \frac{3g}{f} = 3 \cdot g \cdot f^{-1} \in R_{19} \\ &= 3 \cdot (x^5 - x^2 + 1) \cdot (11x^5 + 13x^4 + 4x^3 + 5x^2 + 13) \\ &= 33x^{10} + 39x^9 + 12x^8 - 18x^7 - 39x^6 + 60x^5 + 24x^4 + 12x^3 - 24x^2 + 39 \\ &= 33x^4 \cdot (x^3 - 1) + 39x^3 \cdot (x^3 - 1) + 12x^2 \cdot (x^3 - 1) - 18x \cdot (x^3 - 1) - 39(x^3 - 1) + 60x^5 + 24x^4 + 12x^3 - 24x^2 + 39 \\ &= 15x^5 + 6x^4 + 11x^3 + 2x^2 + 4x + 1 \end{aligned}$$

After this calculation, a secret message m^+ can be chosen:

$$m^+ = x^5 + x^4 + x^2 + x = 011011.$$

We then hash this to obtain the polynomial r, where we take for r the last $n = 6$ output bits of the SHA512 hash:

$$r = \bar{H}(m^+) = 101000.$$

Note that $\bar{H}$ only outputs the last 6 bits for r, in contrary to H which outputs the last 12 bits for both r and the shared secret key K. However, for this example, the secret key K is not of importance and the focus should be on successfully sharing the message m^+, from which the K can always be obtained. Since $r = 101000$, we convert this to a polynomial as

$$r = x^2 + 1.$$

Now we can create m from m^+:

$$m = \mathsf{SOTP}(m^+, G(r)).$$

First, we hash the vector r again, this time using SHA256, and take the last $2n = 12$ bits, which yields

$$G(r) = 111010100110$$

so $u_1 = 111010$ and $u_2 = 100110$. Therefore, we have that

$$\begin{aligned} m &= \mathsf{SOTP}(011011, 111010, 100110) \\ &= (011011 \oplus 111010) - 100110 \\ &= 100001 - 100110 \\ &= (0, 0, 0, -1, -1, 1) \end{aligned}$$

such that we can encrypt m as

$$c = r \cdot h + m \in R_{19}.$$

Hence, we find

$$\begin{aligned} c &= (x^2 + 1) \cdot (15x^5 + 6x^4 + 11x^3 + 2x^2 + 4x + 1) + x^5 - x^4 - x^3 \in R_{19} \\ &= 15x^7 + 6x^6 + 8x^5 + 7x^4 + 14x^3 + 3x^2 + 4x + 1 \\ &= 8x^5 + 3x^4 + x^3 + 3x^2 + 8x + 14 \end{aligned}$$

Finally, we decrypt this ciphertext again:

$$\begin{aligned} m &= f \cdot c \in R_{19} \\ &= (-3x^3 - 3x + 1) \cdot (8x^5 + 3x^4 + x^3 + 3x^2 + 8x + 14) \\ &= -34x^5 - 33x^4 - 77x^3 + 3x^2 - 25x + 41 \\ &= 4x^5 + 5x^4 - x^3 + 3x^2 - 6x + 3 \mod {}^{\pm}19 \end{aligned}$$

And thus, the receiver finds $m = x^5 - x^4 - x^3 \mod {}^{\pm}3$, which is indeed the correct m. Then, we can reconstruct r as

$$\begin{aligned} r' &= (c - m) \cdot h^{-1} \in R_{19} \\ &= (8x^5 + 3x^4 + x^3 + 3x^2 + 8x + 14 - (x^5 - x^4 - x^3)) \cdot (8x^5 + 8x^4 + 7x^3 + 18x^2 + 17x + 11) \\ &= (7x^5 + 4x^4 + 2x^3 + 3x^2 + 8x + 14) \cdot (8x^5 + 8x^4 + 7x^3 + 18x^2 + 17x + 11) \\ &= x^2 + 1 \mod 19 \end{aligned}$$

With this recalculated r', the hash $G(r')$ is recalculated as:

$$G(r') = 111010100110$$

such that $\mathsf{Inv}(m, G(r))$ gives

$$\begin{aligned} m^+ &= \mathsf{Inv}((0,0,0,-1,-1,1), 111010, 100110) \\ &= ((0,0,0,-1,-1,1) + (1,0,0,1,1,0)) \oplus 111010 \\ &= (1,0,0,0,0,1) \oplus 111010 \\ &= 011011 \end{aligned}$$

So this indeed equals the secret m^+. Furthermore, the recalculated r' equals the original r, so the output is validated. Note that we did not calculate the shared secret key K, but this can be obtained from the hash $H(m^+)$, which both parties have access to.

3.5 Similarities and differences

We will now compare the PKE algorithms for NTRU and NTRU+, and show why certain changes were made. First, an overview of the different features of both algorithms is given in Table 1. Note that, since the described instantiations are PKE schemes and not KEMs, the described NTRU+ features will correspond with the algorithms studied in Sections 3.4.4 and 3.4.5.

Feature	NTRU	NTRU+
Ring	$\mathbb{Z}[x]/(x^n - 1)$	$\mathbb{Z}[x]/(x^n - x^{n/2} + 1)$
f	t coefficients 1, $t-1$ coefficients -1	$f = 3f' + 1$, f' coefficients from CBD
g	t coefficients 1 and t coefficients -1	coefficients from CBD
r	same constraints as g	coefficients from CBD
n	not necessarily prime	$2^i 3^j$, $i, j \in \mathbb{N}$ and $i > 0$
q for $\mathbb{Z}_q$	not necessarily prime, but $\gcd(3, q) = 1$	not necessarily prime but $\gcd(3, q) = 1$ To use NTT: q prime and $n \mid (q-1)$
m	coefficients in $\{-1, 0, 1\}$	m^+ coefficients in $\{0, 1\}$
Public key	$h = 3gf^{-1}$	$h = 3gf^{-1}$
Private key	f and its inverse mod 3	f
encryption	$c = r \cdot h + m \in R_q$	$m = \mathsf{SOTP}(m^+, G(r))$ $c = r \cdot h + m \in R_q$
decryption	$a = f \cdot c \mod q^{\pm}$ $m = f_3 \cdot a \mod 3$	$m = f \cdot c \mod {}^{\pm}q \mod {}^{\pm}3$ $r = (c - m) \cdot h^{-1}$ $m^+ = \mathsf{Inv}(m, G(r))$

Table 1: A comparison between NTRU and NTRU+.

3.5.1 Why were these changes made?

As mentioned in Chapter 3.4, NTRU+ has changed the polynomial which defines the ring R to use Number Theoretic Transforms, which speeds up the encryption and decryption process. Although $x^n - x^{n/2} + 1$ is chosen to be an irreducible polynomial over $\mathbb{Z}$, it has many factors in the ring $\mathbb{Z}/q\mathbb{Z}$ where it can be decomposed such that it has factors of at most degree 3.

Moreover, NTRU+ changes the structure of the polynomial f to be $3f' + 1$. This approach saves a step in the decryption: in NTRU, the last decryption step of multiplying a by the inverse of f modulo 3, is done to single out the message m. However, in NTRU+ we have

$$\begin{aligned} f \cdot c &= f \cdot r \cdot h + f \cdot m \mod {}^{\pm}q \\ &= 3g \cdot r + 3f' \cdot m + m \end{aligned}$$

which then yields

$$m \mod {}^{\pm}3.$$

Hence, we do not have to calculate and multiply by the inverse of f in R_3. The downside of this is that the coefficients of the polynomial $f = 3f' + 1$ will be larger, and a larger f will consequently require a larger q to prevent decryption failures from happening, which follows from the explanation in Section 3.1.4.

The way polynomials in NTRU+ are sampled is also different from NTRU. The authors of NTRU+, Kim and Park, use the Centered Binomial Distribution, which has been discussed in Definition 3.1. This sampling originates from a paper by Stehlé and Steinfield from 2011 [37], There, they propose a method to make NTRU provably secure by sampling NTRU polynomials from the Discrete Gaussian Distribution (DGD). By sampling f, g and r from the DGD, they show that the public key h is statistically indistinguishable from being uniformly sampled over its range.

Now, Kim and Park [23] use the fact that the CBD and the DGD are proven to be sufficiently similar, with the help of the Renyí divergence theorem. Hence, we can substitute the DGD by a CBD without introducing any new security concerns. Moreover, some cryptographic systems utilizing Discrete Gaussian sampling have been subject to side-channel attacks, taking advantage of these distributions. An example is the side-channel attack on BLISS, which was proposed by Groot Bruinderink, Hülsing, Lange and Yarom in 2016 [6]. Finally, the CBD is easier to sample than the DGD, as the CBD is a sum of uniform distributions.

Finally, the message space $\mathcal{M}$ in NTRU+ has been changed. Where NTRU accepts polynomials m with coefficients -1, NTRU+ accepts only zero or one as coefficients, hence its name. This change is made to support the ACWC$_2$ algorithm, which NTRU+ introduces to ensure IND-CPA security. Later on, their SOTP produces a new message $m \in \{-1, 0, 1, \}^n$ based on the original m^+.

3.5.2 Keyspace Evaluation

The keyspace is the set of all possible secret keys in a cryptographic system. For a system to be secure, the keyspace is required to be sufficiently large such that a brute force attack on all keys will not easily find the correct secret key. We can test whether a function f is a valid secret key by testing whether $h \cdot f$ has the correct distribution of coefficients. Thus, it is useful to compute an estimate of the keyspace for the different NTRU variants. We will consider the keyspaces of NTRU, NTRU Prime and NTRU+.

In NTRU, the secret key f is taken from the ring R with t coefficients equal to 1 and $t-1$ coefficients equal to -1, while the remaining coefficients are 0. Every polynomial in R has n coefficients, and thus for t ones we have $\binom{n}{t}$ choices. Out of the remaining $n-t$ coefficients, we need $t-1$ to be -1, so there are $\binom{n-t}{t-1}$ choices for this. Now, there are

$$\binom{n}{t}\binom{n-t}{t-1}$$

total choices for f, although this is not yet taking into account that f has to be invertible in R_3 and R_q.

For NTRU Prime, the polynomial f is defined in R with w = 2t nonzero coefficients, and the coefficients in $\{-1, 0, 1\}$, so there are

$$\binom{n}{2t}$$

options for choosing which coefficients will be non-zero. Then for each of these coefficient, it can be either a 1 or a -1, thus 2^{2t} options. In total, this leaves

$$2^{2t}\binom{n}{2t}$$

possible polynomials f.

Finally in NTRU$+$, any polynomial f' of degree at most $n-1$ requires n coefficients. These coefficients can be -1, 0 or 1 according to the CBD. Thus, there are 3^n options for polynomials f', but these are not equally likely. The most likely situation is where $\frac{1}{4}$ of the coefficients are 1, and the same number of coefficients are -1, while the remaining half of the coefficients will be 0.

Too illustrate the relative sizes of these combinatorial expressions, we will calculate the size of n that is required to ensure 256-bit security, for a fixed value of t, in an example.

Example 10. Let us fix the value of $t = 64$. We will calculate the value of n required to obtain 2^{256} different options for the private key. Starting with NTRU, we calculated the size of the keyspace as

$$\binom{n}{t}\binom{n-t}{t-1},$$

so substituting $t = 64$ and equating this to 2^{256} yields the equation

$$\binom{n}{64}\binom{n-64}{63} = 2^{256}.$$

Solving for n then yields $n \approx 169$.

In NTRU Prime, we substitute $t = 64$ in the calculations for the size of the keyspace to obtain

$$2^{128}\binom{n}{128} = 2^{256},$$

which yields a value $n \approx 167$.

For NTRU$+$, there are 3^n options for the polynomial f. Thus, we calculate for which n it holds that

$$3^n = 2^{256},$$

yielding the value $n \approx 162$. Note that a smaller n means that the same level of security can be achieved with a smaller parameter set, therefore causing both ciphertexts and keys to be smaller.

In the above example, it should be noted that all key options in NTRU and NTRU Prime are equally likely, while this is not the case in NTRU+ due to the way f' is generated. Moreover, this example illustrates the required sizes of n to guarantee 256-bit security against brute force attacks. However, stronger attacks are known to exist on NTRU and its variants, and therefore n will generally have to be chosen much larger. This example only serves to illustrate the differences between the required sizes of n for different schemes.

4 Attacks on NTRU variants

Now that the inner workings of several NTRU variants have been discussed, we can begin to look into possible attacks on these systems. We will look at different known attacks on NTRU variants, and see if they apply to the NTRU+ version, taking into consideration the changes discussed in Section 3.4. Moreover, we will show the importance of the SOTP in the NTRU+ system, and explore ways to abuse the SOTP.

We will create proofs of concept for some of these attacks in SageMath [10]. The attacks in this chapter will be split into the categories of brute force and structural attacks. First, however, we must explain why attacks often do not yield the exact private key f, but rather a shifted version.

Attackable Rotations

In any NTRU variant, the public key is always obtained by polynomial division within a ring. Both NTRU and NTRU+ compute the public key h as

$$h = \frac{3g}{f} \text{ in } R_q.$$

For NTRU's ring R, we have already seen in Example 5 that multiplying by the ith power of x results in a shift of the coefficients by i spots. This also means that if we multiply both f and g by the same power of x, these powers cancel and we obtain the same h, since

$$h = \frac{3g}{f} = \frac{3g \cdot x^i}{f \cdot x^i} \quad \forall_{i \in \{0,\dots,n-1\}}.$$

These rotations can therefore also lead to a valid keypair, however we must make a distinction between NTRU and the other discussed NTRU variants here. In NTRU, the quotient polynomial is $x^n - 1$, meaning that multiplying with x is just a convolution, as shown in Example 5. Therefore, these $f \cdot x^i$ have the same number of non-zero coefficients and will not cause extra decryption failures and can each be used for decryption.

However, in NTRU+, the polynomial $x^n - x^{n/2} + 1$ is used as a quotient in the rings. Therefore, when $g \cdot x^i$ or $f \cdot x^i$ have a term of degree larger or equal to n, this term will not simply be shifted, but instead it will be mapped to two different places. Because of this, the $f \cdot x^i$ will no longer have the same number of terms as the original f, once $\deg(f \cdot x^i) \geq n$. A similar problem occurs when $\deg(g \cdot x^i) \geq n$, as the $g \cdot x^i$ no longer contains the same number of nonzero coefficients. This will cause problems in the decryption process, as

$$a = 3g \cdot x^i \cdot r + f \cdot x^i \cdot m$$

may now have coefficients outside the allowed boundaries, thus causing a decryption failure.

In NTRU, there are n options for valid keys, namely each $x^i \cdot f$ for $i \in \{0, \dots, n-1\}$. Any of these keys can be used for decryption, because

$$h = \frac{3g \cdot x^i}{f \cdot x^i}$$

yields the same h. Then, the ciphertext can be decrypted using

$$a = x^i \cdot f \cdot c \mod {}^{\pm}q$$

and then computing

$$m = x^{n-i} \cdot f_3 \cdot a \mod {}^{\pm}3,$$

since x^{n-i} is the multiplicative inverse of x^i in R. If an attacker obtains any of the n valid keys, then the ciphertext can be decrypted, even without the attacker knowing which rotation was originally used to encrypt the message.

In the other NTRU variants, where the quotient polynomial is a trinomial similarly to NTRU+, there are at most a few possible valid keys, depending on the degree of both f and g.

4.1 Brute force attacks

In cryptography, brute force attacks are attacks that rely on trying out all possible options for the private key. Therefore, these attacks rely on the size of the keyspace and the efficiency with which the different keys can be tested. In the following chapters, we will discuss different ways to intelligently try out all options of the secret key.

4.1.1 Meet-in-the-Middle attack

A common attack on PKE systems is the Meet-in-the-Middle (MitM) attack. Here, the keyspace $\mathcal{K}$ is divided in two approximately equally sized subsets. Let us call these two sets K_1 and K_2. The set K is divided as $\mathcal{K} = K_1 \times K_2$ such that

$$|\mathcal{K}| = |K_1| \cdot |K_2|.$$

Then, one of these sets is searched and compared against every entry in the other set. When they satisfy a property specific to the secret key, the two halves are combined and the secret key has been found. It usually provides a trade-off between computation time and memory required, as on average we only have to search through $\sqrt{|\mathcal{K}|}$ secret keys before finding the correct one, but one of the two halves of the keyspace has to be precomputed and stored to compare to the entries from the other half. Moreover, the time it takes to request or look up a stored value should be taken into account. All together, this takes a lot of memory and computational time, which we will quantify in Section 4.1.1.3.

4.1.1.1 Odlyzko's Meet-in-the-Middle attack on NTRU

The instance of MitM that is used on NTRU is known as a variant of Odlyzko's Meet-in-the-Middle attack, which was described in [21] by Howgrave-Graham, Silverman and Whyte in 2003. It splits the secret polynomial f in two parts, which we call f_1 and f_2. Since we do not know what f is, we define two sets S_1 and S_2, such that $f_1 \in S_1$ and $f_2 \in S_2$. Remember that f has t coefficients equal to 1 and $t-1$ equal to -1, with the rest being equal to 0. This MitM search then splits the search space for these nonzero coefficients in the following way: the set S_1 only contains polynomials where the second half of the coefficients are all equal to 0, while the first half consists of all possible combinations of $-1, 0$ and 1. Similarly, the set S_2 contains all polynomials where the first half are only 0 and the second half are the combinations of -1, 0 and 1. Thus,

$$\begin{aligned} S_1 &= \{-1,0,1\}^{\frac{n}{2}} \times \{0\}^{\frac{n}{2}} \\ S_2 &= \{0\}^{\frac{n}{2}} \times \{-1,0,1\}^{\frac{n}{2}} \end{aligned}$$

Thus, we are looking for $f_1 \in S_1$ and $f_2 \in S_2$ such that $f = f_1 + f_2$. Now, we would like a condition which signals that we might have found the correct f_1 and f_2. To do this, the following observation is used:

$$\begin{aligned} h &= \frac{3g}{f_1 + f_2} \\ h \cdot (f_1 + f_2) &= 3g \\ h \cdot f_1 &= 3g - h \cdot f_2 \end{aligned}$$

where we know that g is small by definition. The idea is now to calculate all $h \cdot f_1$ and compare the signs of the coefficients of these polynomials to the coefficients of $-h \cdot f_2$, where the assumption is that $3g$ is so small that it will typically not cause a sign change. To achieve this, we create the sets $h \cdot S_1$ and $-h \cdot S_2$, where we precompute the first and iterate through the latter.

We can assume that $3g$ will not cause a sign change because $h \in R_q$, where q is much larger than 3, hence 3 is small in comparison. So we denote the map Sign, which maps any polynomial with coefficients balanced around q to a tuple containing 1 if the coefficient is positive, and 0 otherwise. Thus:

$$\mathrm{Sign}(f_0 + f_1x + \cdots + f_{n-1}x^{n-1}) = (\mathbb{1}(f_0 > 0), \mathbb{1}(f_1 > 0), \ldots, \mathbb{1}(f_{n-1} > 0)).$$

Now we can compare $\mathrm{Sign}(h \cdot f_1)$ with $\mathrm{Sign}(-h \cdot f_2)$ and if these tuples are equal, we compute a candidate g as

$$g = 3^{-1} \cdot h \cdot (f_1 + f_2)$$

Then simply checking whether all coefficients of this g are in $\{-1, 0, 1\}$ tells us if this g, and hence this $f = f_1 + f_2$, or a rotation of f, are the polynomials that were used to compute h. Thus, we have found the secret key f.

4.1.1.2 Odlyzko's Meet-in-the-Middle attack on NTRU+

When adapting this attack on NTRU+, the main point of concern is the different form of f. We have to adjust the attack such that $f = 3f' + 1 = 3(f_1 + f_2) + 1$, where f_1 and f_2 are the same as before. Then

$$\begin{aligned} h &= \frac{3g}{f} \\ 3g &= h \cdot (3(f_1 + f_2) + 1) \\ 3h \cdot f_1 + h &= 3g - 3h \cdot f_2 \\ h \cdot f_1 + 3^{-1} \cdot h &= g - h \cdot f_2 \end{aligned}$$

We are now able to test $\mathrm{Sign}(h \cdot f_1 + 3^{-1} \cdot h)$ against $\mathrm{Sign}(-h \cdot f_2)$ in the same way as before, so we create the sets $h \cdot S_1 + 3^{-1} \cdot h$ and $-h \cdot S_2$, once again storing the first and iterating through the latter.

Since this is a brute force attack, there is little that can be done to protect the system against it. The best protection is making the attack infeasible by using larger parameters, and thus large keys, such that the time it takes to perform the attack is infeasible.

4.1.1.3 Runtime and Memory

To classify the effectiveness of this attack, we would like to evaluate the runtime of Odlyzko's Meet-in-the-Middle attack on NTRU+. Moreover, we will examine the required amount of memory for this attack to work successfully.

Let us start by calculating the search space S. In NTRU+, the secret key f' contains coefficients in $\{-1, 0, 1\}$ and although unlikely, any of the 3^n options are possible. Therefore,

$$\mathcal{O}(|S|) = \mathcal{O}(3^n).$$

But to run Odlyzko's algorithm, we first have to create the sets containing all options for f_1 and f_2. Each of these sets consists of $3^{n/2}$ elements, as each option has $n/2$ variable entries which can be -1, 0 or 1.

The creation and storage of $h \cdot S_1$, and subsequent iteration through $-h \cdot S_2$, make up the majority of the runtime and memory of Odlyzko's attack. Therefore, the runtime and memory will be of order

$$\mathcal{O}(3^{\frac{n}{2}}) = \mathcal{O}(\sqrt{|S|}).$$

One noteworthy difference in the attack on NTRU and NTRU+ is the comparison of the signs. While in NTRU, this happens on

$$h \cdot f_1 = 3g - h \cdot f_2,$$

this changes into

$$h \cdot f_1 + 3^{-1} \cdot h = g - h \cdot f_2$$

for NTRU+. Thus, in the case of NTRU+, the inverse of $3 \mod q$ has to be calculated, and multiplied with h, before being added to the left-hand side. It is important that the stored set contains the

$$\text{Sign}(h \cdot S_1 + 3^{-1} \cdot h),$$

but the difference with this attack on NTRU is negligible and therefore will barely influence the runtime. Then, Sign$(f_2 \cdot h)$ for $f_2 \in S_2$ will be checked against these stored values.

Note that a simple way to speed up the runtime for Odlyzko's attack on NTRU+ is to start with the most likely candidates for f_1 and f_2 in S_1 and S_2. Because of the Centered Binomial Distribution, secret keys f with half the values ± 1 and half 0 are much more likely to occur than those who are either extremely sparse or dense. Thus, by starting the precomputations for S_1 with polynomials that resemble this CBD distribution, and then continuing with those that deviate slightly more and so forth, and by starting to test elements in S_2 with the most likely distribution, we increase the chance of finding the secret key early. However, this would complicate the process as we would be checking a growing table for each step. Therefore, this analysis only considers the case where we immediately compute S_1 completely.

4.1.2 Golden collision attack

In a standard meet in the middle attack on NTRU, such as the one described in Section 4.1.1, the attacker has to keep track of the sets with all options for f_1. This requires a lot of memory, and therefore, in 2016, a reduced memory Meet in the Middle (MitM) attack on NTRU was proposed by Van Vredendaal [41]. This attack uses the framework from a paper by Van Oorschot and Wiener [40]. They create an algorithm which parallelizes the search for collisions in a semi-random function F on a search space S, i.e., $F : S \to S$. Then, the idea is to start at some $x_0 \in S$ and create trails by running $x_{n+1} = F(x_n)$ until a specific point x_d is reached. Then, these trails are stored by storing their start and end points, and it is checked whether multiple trails end in the same point x_d. If two trails end in the same value, then it is checked if there exist i and j for which $F(x_i) = F(x_j)$ while $x_i \neq x_j$, describing a collision between the two paths. In the following paragraph, we will illustrate how this attack works on NTRU+.

In this attack, we start by splitting the keyspace for f in two groups, similar to Odlyzko's MitM attack. Each of these groups consists of all polynomials of degree $\frac{n}{2}$ with coefficients in $\{-1, 0, 1\}$, which we call x. Moreover, the groups are denoted with a bit b, either 0 or 1, meaning prepending $\frac{n}{2}$ zeros or appending $\frac{n}{2}$ zeros to the coefficient vectors respectively. Together, this set of combinations of a polynomial and a bit are denoted by $S = \{-1, 0, 1\}^{\frac{n}{2}} \times \{0, 1\}$. Note that the actual f is formed by adding two elements (x_1, b_1) and (x_2, b_2) with $b_1 \neq b_2$.

The attack defines a function F which maps from the set S back to S.

$$\begin{aligned} F &: S \to S \\ &(x, b) \mapsto (x, b) \end{aligned}$$

The function F performs the following transformations consecutively: it takes an element in S, so a bit $b \in \{0, 1\}$ and a vector $x \in \{-1, 0, 1\}^{\frac{n}{2}}$ and constructs the vector v:

$$v = \begin{cases} 00\ldots0 \| x & \text{if } b = 0 \\ x \| 00\ldots0 & \text{if } b = 1 \end{cases}$$

where $00\ldots0$ are $\frac{n}{2}$ zeros, to make v a vector in $\{-1, 0, 1\}^n$. Now, the function continues by multiplying this vector of length n by h and adding $3^{-1} \cdot h$ if $b = 0$, or multiplying by $-h$ if $b = 1$. Next, it converts the obtained polynomial to another vector of length n, which we call v_b, by taking the Sign function of each coefficient. So we calculate Sign$(h \cdot v + 3^{-1} \cdot h)$ when $b = 0$ or Sign$(-h \cdot v)$ when $b = 1$ Finally, this vector v_b is hashed, together with a random element e in $\{0, 1, \ldots, 2^{64}\}$. Note that this e is not chosen randomly every time v_b is hashed. Instead, it is randomly generated once and stored to be used for a longer period of time. Only when it becomes apparent that the algorithm will not terminate with the current distribution of points, will the random element e be updated.

Then, this hash function, in our case chosen to be hashed with SHA512, of v_b and e, yields a 512 bit output. In any case, the output must be larger than $n + 2$ bits. From this output, the second most significant bit is then taken as the new bit, while we obtain a new vector $x \in \{-1, 0, 1\}^{\frac{n}{2}}$ by taking the last n bits. This binary number is then mapped to a specific polynomial in $\{-1, 0, 1\}^{\frac{n}{2}}$ using some deterministic function, which we call "Detcomb":

$$\text{Detcomb} : \{0, 1\}^n \to \{-1, 0, 1\}^{\frac{n}{2}}.$$

This function deterministically computes a new element of S as follows: it splits the first n bits of this input into two groups of $n/2$ bits. It then uses these groups to sample a new vector x by applying the CBD ψ_1. Then, we have found our way back to another bit and a vector of length $\frac{n}{2}$ with entries in $\{-1, 0, 1\}$, hence this is an element of S and the description of the function F is complete. This attack aims to find collisions within in this function F, similar to Odlyzko's MitM attack. Note, however, that we require a collision from a left half and a right half, matching a value with $b = 1$ and one with $b = 0$. So only some collisions will turn out to be useful, which are then referred to as golden collisions.

To create the table of trails, we need a way to define specific elements of S. These elements are called distinguished points, and we let the set of all distinguished points be D. An element in S sits in D if the last 32 bits of the SHA512 hash of the length $\frac{n}{2}$ vector x together with the random element e is less than $2^{32} \cdot \theta$ where θ is the desired ratio of distinguished points. To randomize the set of distinguished points further, and to reshuffle them when e changes, we also input the e to this hash function, as proposed by Cottaar [8].

Hereafter, so-called trails are found by starting at a random element $x_0 \in S$ and applying F to S until we reach a point in D. While creating these trails, we keep track of the starting point and the number of steps it takes until we reach our distinguished point. Then, we add all the information about the trail to a dictionary. We require a mechanism to efficiently look up elements in this dictionary, which allows us to check whether we reached the same distinguished point. Thus, we index the dictionary by the integer representing the last k bits of the distinguished point that was reached, while the value is formed by a tuple of the starting vector and bit, and the number of steps the trail took before reaching the distinguished point. Note that the index being a hash of the distinguished point limits the number of different trails that will be stored, as different distinguished points may have the same representation if we choose the k small enough.

We keep adding trails to this dictionary until there is a match in the indices. If this occurs, we might have found our golden collision. First, we have to check that the shorter trail is not part of the longer trail. This is done by running the longest of the two trails until they are both the same length. If these are now both in the same element of S, we have a so called Robin-Hood collision. We then simply keep the longest of the two trails in the dictionary, while discarding the other.

Otherwise, there is still a possibility that these trails will lead to a collision, and we continue as follows. From this point onward, both trails are equal length. We now want to find the two elements in S that map to the same point, but are not equal. So we iterate through the trail, while keeping track of the previous point, until we hit the first point that occurs in both trails, i.e., until we find the tuples (x_1, b_1) and (x_2, b_2), with $x_1, x_2 \in \{-1, 0, 1\}^{\frac{n}{2}}$, such that $F(x_1, b_1) = F(x_2, b_2)$. In Figure 4, the points in green, x_3 and y_2, both map to the same point. Therefore, we are interested in x_3 and y_2 as they form a potential golden collision.

Note that we can only reconstruct a candidate f if we have a first and second half polynomial, hence we need that $b_1 \neq b_2$. Then, we continue by creating the possible candidate for f. We convert the vectors x_1 and x_2 to a length n vector by appending or prepending $\frac{n}{2}$ zeros, based on b_1 and b_2 respectively. Now, we proceed as we

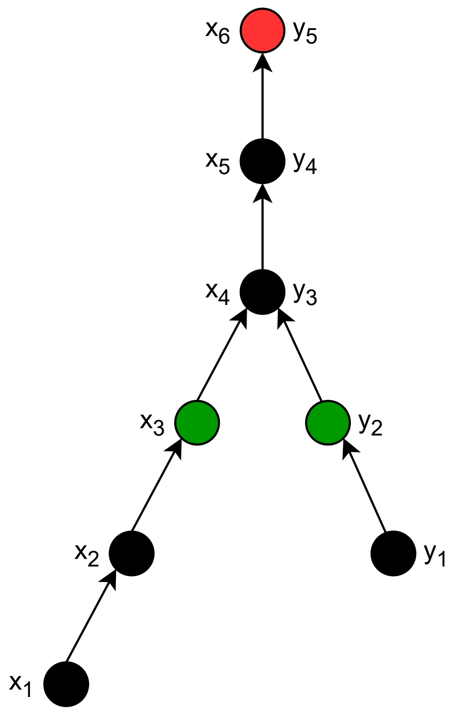

Figure 4: Two colliding paths leading to the same distinguished point

did with the half polynomials f_1 and f_2 in Odlyzko's attack, by writing:

$$3h \cdot f_1 + h = 3g - 3h \cdot f_2,$$
$$h \cdot f_1 + 3^{-1} \cdot h = g - h \cdot f_2$$

we check whether the vectors $3h \cdot f_1 + h$ and $-3h \cdot f_2$ have the same Sign. Here, the assumption that $3g$ will not change the signs is made. Finally, g is again computed as

$$g = \frac{(3 \cdot (f_1 + f_2) + 1) \cdot h}{3}$$

and if this vector is ternary, we have found the correct f and g.

If the correct f is not found after finding a specific number of trails, we update the random element e. This changes both the distinguished points as well as the hash in the function F, thus reshuffling the whole search space. Moreover, if a trail exceeds a certain number of steps, we will start a new trail at a random point in S, as we are likely stuck in a loop. For example, if we observe the graph in Figure 5, we notice a loop between two of the points. If we were to randomly select either of these points, the trail will never hit a distinguished point. However, the graph does contain some possible collision, and perhaps even the golden collision. Thus, instead of choosing a new e and switching up the whole graph, we only select a new random point to start iterating from.

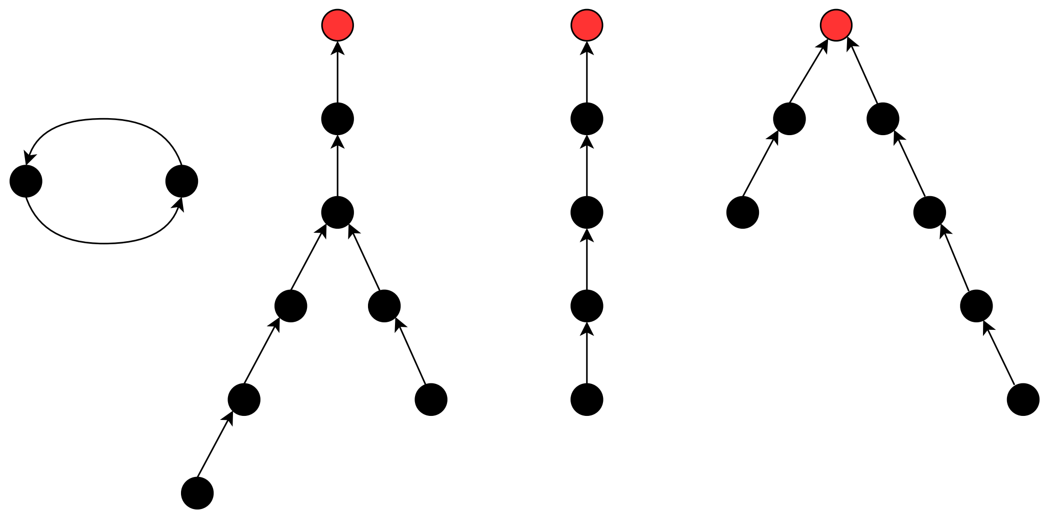

Figure 5: A graph containing a loop without any distinguished points

4.1.2.1 Runtime and memory

The golden collision attack is an attack specifically designed to lower the memory requirements of Odlyzko's Meet-in-the-Middle attack. However, for lower memory usage, it sacrifices the runtime. The ideal feature of this attack is that the amount of memory can be allocated by the attacker himself, depending on how much memory is available or how much the attacker wants to spend. This is done by setting the output length of the hash function which stores the trail based on its distinguished point. If this output is chosen to be small, many distinguished points will be hashed to the same value, thus keeping the list of trails that has to be stored small. This will cause the algorithm to terminate much slower, as a collision is far less likely to emerge from two trails hitting the same distinguished point, and instead often arises from two distinct distinguished points hashing to the same value.

An interesting statistic is the expected time to find the golden collision. According to Van Oorschot and Wiener [40], the expected time to find a collision is

$$\sqrt{\frac{|S| \cdot \pi}{2}}.$$

Moreover, Van Oorschot and Wiener show that the runtime for a golden collision attack is

$$\mathcal{O}\left(\frac{|S|^{\frac{3}{2}}}{\sqrt{w}}\right),$$

where w is the available amount of available memory, i.e, the number of trails that can be stored in the hash list and thus the size of the image of the hash function.

4.1.3 Meet-LWE

We have seen multiple different brute force attacks on the NTRU system and its variants. A more recent attack, which offers a significant speed up, was proposed in 2021 by May [28]. The attack, called Meet-LWE, is specifically designed to attack Learning With Errors (LWE) systems of linear equations of the form

$$As = b + e \mod q \tag{5}$$

where $A \in \mathbb{Z}_q^{n\times n}$ and $b \in \mathbb{Z}_q^n$ are known components, while $s \in \mathbb{Z}_q^n$ and $e \in \{0,1\}^n$ are secret. However, May's algorithm will take $s, e \in \{-1,0,1\}^n$, which will allow us to adapt this attack on NTRU and NTRU+.

The algorithm proposed by May starts by transforming the LWE secrets into two halves, such that $s = s_1 + s_2$ and $e = e_2 - e_1$, similar to the approach taken in Odlyzko's MitM attack:

$$A(s_1 + s_2) = b + e_2 - e_1$$
$$As_1 + e_1 = b - As_2 + e_2$$

where $e_1 \in \{-1,0,1\}^{\frac{n}{2}} \times \{0\}^{\frac{n}{2}}$ and $e_2 \in \{0\}^{\frac{n}{2}} \times \{-1,0,1\}^{\frac{n}{2}}$. For more detail on the split of s, we refer to Section 6 to 8 in the paper by May [28]. In particular, there are D different representations for each s. Now, a random target vector $t \in \mathbb{Z}_q^d$ is chosen, where $d = \lfloor \log_q(D) \rfloor$.

Then, May defines a projection π_d which maps any vector of length n to just its first d values:

$$\pi_d : \mathbb{Z}_q^n \to \mathbb{Z}_q^d$$
$$x = (x_1, \ldots, x_n) \to (x_1, \ldots, x_d).$$

Since the output $\mathbb{Z}_q^d$ of π_d is $q^d < D$, we expect that there is at least one of the D representations (s_1, s_2) of s, such that the projection

$$\pi_d(As_1 + e_1) = t \mod q,$$

but then

$$\pi_d(b - As_2 + e_2) = t \mod q$$

too. Now, lists are constructed from vectors that solve these equations:

$$L_1 = \{(s_1, l(As_1)) \mid \pi_d(As_1 + e_1) = t \mod q\},$$
$$L_2 = \{(s_2, l(b - As_2)) \mid \pi_d(b - As_2 + e_2) = t \mod q\},$$

where l is a hash function

$$l : \mathbb{Z}_q^n \to \{0,1\}^n \text{ with } l(x)_i = \begin{cases} 0 & \text{if } 0 \le x_i < \lfloor \frac{q}{2} \rfloor - 1 \\ 1 & \text{if } \lfloor \frac{q}{2} \rfloor \le x_i < q - 1 \end{cases}.$$

The border values $\lfloor \frac{q}{2} \rfloor - 1$ and $q - 1$ will be assigned both a 0 and 1, as these values in e_1 or e_2 can result in a flip of the hash value. For a more in depth explanation, we refer to the paper by May [28].

Finally, for all matches $(s_1, \cdot)$ and $(s_2, \cdot)$ in the second component of $L_1 \times L_2$ we check whether $s = s_1 + s_2 \in \{-1,0,1\}^n$ and $(As - b) \mod q \in \{-1,0,1\}^n$. If this is the case, we have found the secret s.

4.1.3.1 Applying Meet-LWE to NTRU+

One instance in NTRU and NTRU+, where this algorithm is directly applicable, is on the encryption or encapsulation step

$$c = h \cdot r + m.$$

Note that, since this step happens in a ring R_q, we can view $h \cdot r$ as a multiplication of a vector r by a matrix H. Then, rewriting this to the form

$$H \cdot r = c - m$$

lets us recognize the form of an LWE system immediately: We have $H \in \mathbb{Z}_q^{n\times n}$ and $c \in \mathbb{Z}_q^n$ as known components, while $r \in \{-1,0,1\}^n$ and $m \in \{-1,0,1\}^n$ are unknown. Thus, splitting $r = r_1 + r_2$ and $m = m_1 - m_2$, with $m_1 \in \{-1,0,1\}^{\frac{n}{2}} \times \{0\}^{\frac{n}{2}}$ and $m_2 \in \{0\}^{\frac{n}{2}} \times \{-1,0,1\}^{\frac{n}{2}}$, yields the system

$$H \cdot r_1 + m_1 = c - H \cdot r_2 + m_2.$$

On this system, Meet-LWE can be applied directly. However, applying a successful Meet-LWE on this system will only yield the secret message and polynomial r, but not the secret key.

Another equation where Meet-LWE can be applied, is on the NTRU+ key generation. Recall that the public key h is calculated as

$$h = \frac{3g}{f} \text{ in } R_q$$

where both g and f have coefficients in $\{-1,0,1\}$ and are secret. Thus, we can write this as

$$\begin{aligned} h \cdot f &= 3g \text{ in } R_q \\ 3^{-1} \cdot h \cdot f &= g \text{ in } R_q \end{aligned}$$

Now, note that we can consider multiplying f by $3^{-1} \cdot h$ a matrix multiplication in R_q, and let H be this matrix. This yields the system

$$H \cdot f = g \mod q,$$

which is an LWE system on which Meet-LWE is applicable, because $H \in \mathbb{Z}_q^{n\times n}$, $f \in \{-1,0,1\}^n$ and $g \in \{-1,0,1\}^n$. Note that in NTRU+,

$$f = 3f' + 1 = 3(f_1' + f_2') + 1$$

so by substituting this and $g = g_2 - g_1$, we arrive at

$$H \cdot (3(f_1' + f_2') + 1) = g_2 - g_1 \tag{6}$$
$$3 \cdot H \cdot f_1' + g_1 = -h - 3 \cdot H \cdot f_2' + g_2. \tag{7}$$

Note that we have defined a multiplication with $3^{-1} \cdot h$ in R_q as H, however an addition with the polynomial h is still simply a vector addition. Therefore, $-h$ now takes the role of the known vector b in the classical LWE system of Equation 5. We can now apply Meet-LWE to Equation 6 to directly target the private key.

4.1.3.2 Runtime and memory

In the Meet-LWE algorithm, the runtime and memory, like in Odlyzko's attack, are dominated by the size of the lists. Although the calculations and tests fall outside the scope of this thesis, we will briefly mention the most important results from May's analysis. For a more in-depth analysis, we refer to the paper by May [28].

The guessing of the first d components of e, as achieved by the function π_d, takes a complexity of

$$\mathcal{O}\left(\frac{n}{\log n}\right).$$

Under asymptotic optimization, the second instantiation of the algorithm proposed by May, called Rep-1, achieves list construction in

$$\mathcal{O}(|S|^{\frac{1}{4}}).$$

Thus, this algorithm performs significantly better than Odlyzko's algorithm.

4.2 Structural attacks

Structural attacks aim to exploit vulnerabilities in the design or implementation of a cryptographic system. For NTRU and its variants, we will attempt to abuse the underlying mathematical structure of the lattice and the ring R to find the secret key or the secret message m. In the case of NTRU+, we will also be considering the proposed SOTP as a potential attack vector.

4.2.1 Evaluation-at-one attack

The evaluation-at-one attack is one of the simplest attack strategies on the NTRU system. It does not fully break security, but rather reveals some information about the used parameters.

4.2.1.1 Evaluation-at-one on NTRU

In classic NTRU, the polynomial f is defined as having t coefficients equal to 1 and $t-1$ equal -1. Moreover, g is defined with both t coefficients equal to 1, and also t coefficients equal to -1. Now, we know that the public key

$$h = \frac{3g}{f}.$$

Moreover, the encryption process encrypts a message m as

$$c = r \cdot h + m$$

Now, r is a polynomial with the same constraints as g, so defined with both t coefficients equal to 1, and also t coefficients equal to -1. Therefore, $h(1) = \frac{3 \cdot g(1)}{f(1)} = \frac{3 \cdot 0}{1} = 0$ and $r(1) = 0$. Hence, evaluating the ciphertext c in the point $x = 1$ will yield

$$c(1) = r(1) \cdot h(1) + m(1) = m(1)$$

so the message m evaluated at 1. Now this does not immediately render the system useless, but it does give some information about the secret m, and is therefore undesirable.

4.2.1.2 Evaluation-at-one on NTRU+

To prevent these attacks from happening, a solution is to change the polynomial ideal of the ring R in which we work, such that 1 is no longer a root. However, this approach will still cause $\frac{100}{q}\%$ of the message and their corresponding ciphertexts to have the same value when evaluated at 1. This is because there are q options for $m(1) \mod q$ and $c(1) \mod q$. Thus, there is a $1/q$ probability that these evaluations are still equal. Furthermore, one can find other roots ξ of the quotient ring of R to evaluate the polynomials at, for which the probability that $m(\xi) = c(\xi)$ is significantly higher than $1/q$. To randomize even further, some variants of NTRU, such as NTRU+, do not use a fixed number of coefficients equal to 1 or -1 in the generated polynomials, but instead they draw them from a random distribution. Therefore, the evaluation-at-one attack is not viable on NTRU+.

4.2.2 LLL lattice attacks

As we have seen in Section 3.1.6, NTRU is a lattice-based cryptographic system. These lattices present us with a possible way to attack the system. We have already shown that the pair $\begin{pmatrix} g & f \end{pmatrix}$ is one of the shortest vectors in the NTRU lattice. Furthermore, Coppersmith and Shamir showed in 1997 [7] that we do not necessarily need the shortest vector to decrypt with . Let n_f the length of the vector f, and let $n_{f'}$ the length of alternative vectors for decryption f'. If we manage to find two vectors f'^1 and f'^2 such that

$$n_{f'^1} = n_{f'^2} \leq 2.5 \cdot n_f,$$

we can obtain partial information, which can be combined to obtain m. Finally, if we obtain several f'^i with

$$n_{f'^i} \approx 4 \cdot n_f,$$

we can apply error correcting techniques to still obtain the m. However, algorithms such as the Lenstra-Lenstra-Lovász algorithm, which has been discussed in Section 2.5.1, can find a somewhat shorter vector in such a lattice efficiently, but for large enough n, they will never yield a vector of this length.

4.2.2.1 LLL attack against NTRU

Recall that the lattice for a standard NTRU system is given by

$$\begin{bmatrix} qI_n & 0 \\ H & I_n \end{bmatrix}$$

where H is equal to polynomial multiplication with $\frac{h}{3} \mod q$. Thus, this whole lattice is public and anyone has access to it.

This makes it seem as if NTRU is not very secure. However, as Hoffstein, Pipher and Silverman point out [18], if n is sufficiently large, a much stronger reduction algorithm is needed. For these large n, the LLL algorithm failed to terminate, which the authors claim is likely due to round-off errors. Using modern implementations of the LLL algorithm should prevent these overflow issues, however the approximation factor in LLL is too large to give short enough vectors to be used for correct decryption.

4.2.2.2 LLL attack against NTRU+

Moreover, if we want to expand this attack to NTRU+, we first note that key generation is largely equal to normal NTRU, except for the used rings. Therefore, the lattice

$$\begin{bmatrix} qI_n & 0 \\ H & I_n \end{bmatrix}$$

can be reused. But note that H will not be a simple shift of the multiplications of $\frac{h}{3}$, but instead the coefficients will jump around because of the ideal $x^n - x^{n/2} + 1$. Therefore, a lattice basis reduction algorithm, such as LLL, is very unlikely to find the vector $\begin{pmatrix} g & f \end{pmatrix}$, or an equivalent key. Instead, for large enough n, the LLL algorithm will find much longer vectors in the lattice, which cannot be used for decryption.

4.2.3 BKZ lattice reduction

Much like LLL discussed in Section 4.2.2, another attack exist using the Block Korkine-Zolotarev (BKZ) algorithm. This algorithm is based on ideas from a 1877-paper by Korkine and Zolotareff [24], but it was created by Schnorr in 1987 [33]. Like the LLL algorithm, it aims to reduce the basis of a lattice. In fact, the BKZ algorithm starts by first running a few cycles of LLL to somewhat shorten the basis of a given lattice. Then, it splits this shorter basis into blocks, each consisting of β basis vectors. Next, it runs some lattice attacks on the smaller blocks iteratively. There are exponential-time computations to solve SVP and return shortest vectors in the smaller blocks. There are two types of algorithms for this computation, called enumeration and sieving, with sieving being asymptotically faster. Finally it merges the reduced blocks to reconstruct a shortened basis of the full lattice. One can then repeat this process with larger block sizes to continue shortening the basis.

In a paper from 2018, Albrecht, Curtis, Deo, Davidson, Player, Postlethwaite, Virdia and Wunderer provide code to estimate the security of any Learning With Errors (LWE) or NTRU scheme [2]. They apply the BKZ algorithm on these schemes, and they predict the running time using several available models. In literature, the cost of finding a shortest vector in a lattice of dimension β using sieving is usually estimated as

$$2^{c \cdot \beta + o(\beta)}.$$

The value of c is classically 0.292, as proposed by Becker, Ducas, Gama and Laarhoven in 2016 [4]. However, other algorithms exist that achieve smaller constants, such as the quantum Grover speed-up, which yields $c = 0.265$.

We estimate the security of NTRU+ using the code by Albrecht, Curtis, Deo, Davidson, Player, Postlethwaite, Virdia and Wunderer. However, to do this, some assumptions had to be made, as the code was not fully applicable to the NTRU+ system. First, it required the exact number of coefficients equal to 1 and -1 in the generated keys f, g and r. Because NTRU+ uses the Centered Binomial Distribution from Definition 3.1, the exact number of nonzero coefficients varies. Thus, we assumed that the system would generate exactly $\frac{1}{4}$ of the coefficients as 1, and a further $\frac{1}{4}$ as -1, following the expectation of the CBD.

Next, a set value for the 2-norm of the private key, so $\|f\|_2$, is also required. Here, we have to consider that $f = 3f' + 1$ and that f' is generated according to the CBD. Thus, again assuming that NTRU+ will generate keys with exactly $\frac{1}{4}$ of the coefficients as 1 and $\frac{1}{4}$ as -1, we take

$$\|f\|_2 = \|3f' + 1\| \approx \|3f'\| = \sqrt{9 \cdot \frac{n}{2}}.$$

With the above assumptions, the estimated security levels are calculated for the 4 suggested parameter sets of NTRU+. Although the explanation of dual attacks falls outside the scope of this thesis, the results for dual attacks are shown in Table 2, using the code provided by the authors of the paper. Note that the first column contains the asymptotic estimates used for reduction in dimension β, and that these estimates are $c\beta + o(1)$ in the exponent of the runtime. Also observe that many different options for the cost of the quantum Grover speed-up subroutine exist, which were all incorporated in the table for the sake of completeness.

n q	576 3457	768 3457	864 3457	1152 3457
Proposed BKZ cost models				
0.265β	104	149	171	273
0.265β+16.4	120	165	187	289
0.2975β	117	167	192	306
0.265β + $\log(\beta)$	113	158	180	283
0.265β +16.4 + $\log(8\text{d})$	134	178	201	303
0.292β	115	164	189	300
0.292β + 16.4	131	180	205	316
0.368β	144	206	237	378
0.292β + $\log(\beta)$	123	173	198	310
$0.292\beta + 16.4 + \log(8d)$	144	194	219	331
$\frac{1}{2}(0.187\beta\log(\beta) - 1.019\beta + 16.1)$	124	200	242	447
$0.125\beta\log(\beta) - 0.755\beta + 2.25$	127	216	264	508
$0.187\beta\log(\beta) - 1.019\beta + 16.1$	248	399	482	887
$0.000784\beta^2 + 0.366\beta - 0.9 + \log(8d)$	276	460	564	1071

Table 2: Security estimates for different proposed NTRU+ parameters

These values can be compared to the claimed security of dual attacks on NTRU+, which can be seen in Table 3. These results are obtained from the script used by Kyber, an LWE-based KEM introduced by Avanzi, Schwabe, Bos, Ducas, Kiltz, Lepoint, Lyubashevsky, Schanck, Schwabe, Seiler and Stehlé in 2017, and later updated for each round of the NIST competition [34].

n	576	768	864	1152
q	3457	3457	3457	3457
0.265β	104	146	171	240
0.292β	115	161	188	264

Table 3: Claimed security levels for different proposed NTRU+ parameters

Note that for similar block sizes and parameters, the estimated security is very similar to the claimed security. Only when $n = 1152$ do the values differ by more than 3 bits, but the claimed security values are always lower meaning that the code used by the NTRU+ authors is stricter than the one we used to estimate security levels.

4.3 Reaction attacks

A reaction attack is an attack that observes the reaction of a cryptographic system to certain inputs, which are chosen by the attacker. By checking which inputs take longer to decrypt, consume more power, or yield decryption failures, the attacker can deduce information about the private key of the system. This type of attack was first shown against various different lattice-based PKE systems by Hall, Goldberg and Schneier in 1999 [17]. However, NTRU was not among the systems that got attacked in this initial paper. Therefore, a technical report was written by Hoffstein and Silverman in 2000, where they discuss a reaction attack designed for NTRU [19].

4.3.1 Reaction attack on NTRU

Recall from Section 3.1.4 that decryption failures can happen if q is not chosen such that

$$\frac{q}{2} > 4d,$$

where d is the maximum number of non-zero coefficients in r, g, f and m. Now, let us assume there is an oracle which, upon receiving c, outputs that a decryption failure occured if a reduction modulo q took place while calculating a. Furthermore, assume that q is not chosen in the way described above, so decryption failures can occur.

If an attacker obtains a valid ciphertext c, they can check the validity by sending c to this oracle. If c is a correctly generated ciphertext, the oracle should not output anything. Now, this attacker can slightly modify c as $c' = c+1$, so the decrypter will find

$$a' = f \cdot (c+1) = f \cdot c + f = a + f = \sum_{i=0}^{n-1} (a_i + f_i) \cdot x^i$$

Since a computed from c did not yield a decryption failure, we know that if a' causes a decryption failure, then one of the a_i is $\frac{q}{2}$ and the corresponding f_i is 1. Similarly, trying $c' = c - 1$ yields a decryption error if and only if $a_i = \frac{q}{2}$ and $f_i = -1$.

We can extend this further, taking

$$c' = c + l \cdot x^k$$

which yields

$$a' = a + l \cdot f \cdot x^k.$$

Then, for all $l \in \{1, q-1\}$, we can simply try all $k \in \{0, n-1\}$ until the first decryption failure is detected. When this happens, we take this l and try $c' = c + l \cdot x^k$ and $c' = c - l \cdot x^k$ for all $k \in \{0, n-1\}$. Let $k_0^+, k_1^+, \dots, k_{n-1}^+$

denote all k for which $c' = c + l \cdot x^k$ yields a decryption failure, and similarly let $k_0^-, k_1^-, \dots, k_{n-1}^-$ all k for which $c' = c - l \cdot x^k$ gives a decryption failure. Then some rotation of f can be found by calculating

$$f = \left(x^{n-k_0^+} + x^{n-k_1^+} + \dots + x^{n-k_{n-1}^+}\right) + x^j \left(-x^{n-k_0^-} - x^{n-k_1^-} - \dots - x^{n-k_{n-1}^-}\right)$$

for all $j \in \{0, 1, \dots, n-1\}$. From this f candidate we can compute a g as

$$g = 3^{-1} \cdot f \cdot h \mod {}^{\pm}q \in R_q$$

and we check whether g only has coefficients in $\{-1, 0, 1\}$. Moreover, f can be used to calculate f_3, which together with f can be used to decrypt ciphertexts c.

Note that, since q is often odd, the interval in which coefficients of a can lay is then $\left(-\frac{q}{2}, \frac{q}{2}\right]$. This means that adding 1 to cause an overflow, or subtracting 1 to cause an underflow is not distinguishable. This will slightly complicate the process.

4.3.2 Reaction attack on NTRU+ without SOTP

One of the notable differences in NTRU+ is the fact that they first generate a message $m^+ \in \{0,1\}^n$, which is changed to a message $m \in \{-1,0,1\}^n$ by the SOTP. First, let us look at what happens when this SOTP transformation never happens, and the message $m^+ \in \{0,1\}^n$ is encrypted directly via

$$c = h \cdot r + m^+ \text{ in } R_q.$$

Then, the decapsulation process would work as

$$m^+ = f \cdot c \mod {}^{\pm}q \mod {}^{\pm}3,$$

where we do not need to multiply by f_3 because $f = 3f' + 1$. This decryption will yield an m^+ only containing coefficients in $\{0,1\}$, even though the $\mod {}^{\pm}3$ could give outputs with a coefficient -1. In other words, the actual output space and the desired output space do not match. Assume that we have access to a decryption oracle which outputs a decryption failure if the computed m contains at least one coefficient that is equal to -1.

With this in mind, an attacker can modify a valid ciphertext c as $c' = c + x^k$ for all $k \in \{0, \dots, n-1\}$, and send this to the oracle. This gives

$$\begin{aligned} m' = f \cdot c' &= 3g \cdot r + f \cdot m + f \cdot x^k \mod {}^{\pm}q \\ &= 3g \cdot r + 3f' \cdot m + 3f' \cdot x^k + m + x^k, \end{aligned}$$

which then yields

$$= m + x^k \mod {}^{\pm}3.$$

Thus, this can yield a decryption failure if the k-th coefficient of m, say m_k, equals 1. However, it is also possible that, due to the addition of the $3f' \cdot x^k$, this will exceed the boundaries $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$, and cause a reduction modulo q. Therefore, this might give away some information about the secret key f, however it will also obstruct the message recovery process.

Assuming that the $3f' \cdot x^k$ will not cause a reduction modulo q, then the decryption failure only happens if the k-th coefficient of m, say m_k, equals 1. Thus, we let $k_0, k_1, \dots, k_{n-1}$ the k's for which $c' = c + x^k$ yields a decryption failure, then

$$m^+ = x^{k_0} + x^{k_1} + \dots + x^{k_{n-1}}.$$

4.3.3 Reaction attack on NTRU+

To sample a message with an arbitrary distribution in $\{-1,0,1\}^n$ from a message in $\{0,1\}$, the authors of NTRU+ implement the SOTP. First, we will attempt to set up a generic reaction attack against the full NTRU+ KEM. Therefore, we once again assume we have a valid c, and edit it as

$$c' = c + l \cdot x^k.$$

Decryption by an oracle then yields

$$\begin{aligned} a' &= f \cdot c' \mod {}^{\pm}q \\ &= f \cdot (c + l \cdot x^k) \mod {}^{\pm}q \\ &= a + l \cdot f \cdot x^k. \end{aligned}$$

So, this forged ciphertext produces the exact same intermediate value as in a reaction attack on classic NTRU. However, the difference with normal NTRU is that $f \cdot x^k$ will not simply shift, but terms will also appear in the middle of this expression because

$$R_q = (\mathbb{Z}/q\mathbb{Z})[x]/(x^n - x^{n/2} + 1).$$

maps x^i to $x^{i/2} - 1$ when $i \geq n$. Therefore, we cannot simply use the same tactic as in NTRU, because decryption failures can occur at multiple different coefficients of a' depending on the size of k. In other words, like in the previous part, we cannot pinpoint the decryption failure that happens at k^+ or k^- to a term x^{n-k^+} or x^{n-k^-} in f.

Something that is worth researching further, is whether the values of k can still reveal some information about the private key f. Possibly, we can attempt to combine different values of l and k, for which a decryption failure happens, using combinatorics or linear algebra. However, since q is odd, the range in which the coefficients a_i of a should lay is $\left[-\frac{q-1}{2}, \frac{q-1}{2}\right]$. Because of the symmetry of this interval, it is not certain whether a potential decryption failure is caused by an overflow or an underflow of this interval. Assuming that there is a single largest coefficient a_i and because (g,f) and $(-g,-f)$ are equivalent, this does not cause further complications in the attack.

4.3.4 Reaction attack on SOTP

One attack vector which we have not yet discussed is the Semi-generalized One Time Pad (SOTP) in the NTRU+ KEM. First, we will consider an attack proposed by Lee in the Korean Post-Quantum Competition bulletin board in 2023 [25]. This attack is based on the implementation of NTRU+ as given by the authors, and the vulnerable mechanics are not mentioned in the paper. Recall the SOTP as described in Section 3.4.2, consisting of the function

$$\begin{aligned} \text{SOTP} : \{0,1\}^n \times \{0,1\}^n \times \{0,1\}^n &\to \{-1,0,1\}^n \\ (x, u_1, u_2) &\to y = (x \oplus u_1) - u_2 \end{aligned}$$

and its inverse

$$\begin{aligned} \text{Inv} : \{-1,0,1\}^n \times \{0,1\}^n \times \{0,1\}^n &\to \{0,1\}^n \\ (y, u_1, u_2) &\to x = (y + u_2) \oplus u_1. \end{aligned}$$

Also recall from Section 3.4 that the Inv function takes the input $(m, G(r))$ in the NTRU+ KEM, where

$$m = f \cdot c \mod {}^{\pm}q \mod {}^{\pm}3.$$

However, note that editing the ciphertext c also influences the plaintext m, which is then injected into the Inv function. Note that, in the Inv function,

$$(y + u_2) = (x \oplus u_1) - u_2 + u_2 = x \oplus u_1$$

so this term will only contain zeros and ones such that it can be XOR-ed with u_1. But now if m is changed, where m takes the form of y in the expression of the SOTP and the Inv above, then $y + u_2$ may no longer be binary. The NTRU+ implementation takes each entry of the vector $y + u_2$ modulo 2, i.e, it takes only the least significant bit. But then, Lee proposes to change the ciphertext as

$$c' = c + 2$$

such that

$$\begin{aligned} m' &= f \cdot c' \mod {}^{\pm}q \mod {}^{\pm}3 \\ &= f \cdot (c + 2) \\ &= f \cdot c + 2 \cdot f \\ &= 3g \cdot r + f \cdot m + 2 \cdot 3f' + 2, \end{aligned}$$

which becomes

$$m' = m + 2 \mod {}^{\pm}3.$$

However, note that this only works when the extra term $2 \cdot 3f'$ does not overflow a coefficient somewhere, which would trigger a reduction modulo q. Fortunately, the suggested parameter sets for NTRU+ all contain q that are large enough to normally prevent decryption failures from happening. Moreover, if $m[0]$ is not equal to -1, then adding 2 will cause a reduction mod ${}^{\pm}3$ and thus the resulting m' will not equal $m + 2$, which will cause $r' \neq r$.

First note that the decapsulation of r, as shown in Section 3.4.7, gives:

$$\begin{aligned} r &= (c' - m') \cdot h^{-1} \\ &= (c + 2 - (m + 2) \cdot h^{-1} \\ &= (c - m) \cdot h^{-1} \end{aligned} \tag{8}$$

so the r does not change when the ciphertext is modified by addition or subtraction, as long as the reductions modulo q and 3 are not triggered by the extra terms. Therefore, u_1 and u_2 are the same as they were in the SOTP because $u_1 \| u_2 = G(r')$ and $r' = r$. Then,

$$\begin{aligned} \mathsf{Inv}(m', u_1, u_2) &= (m' + u_2) \oplus u_1 \\ &= (m + 2 + u_2) \oplus u_1 \\ &= (m + u_2) \oplus u_1 \\ &= m^+ \end{aligned}$$

where the 2 is dropped because of the modulo 2 in the implementation of NTRU+. However, Lee claims that this idea will only be effective when $m[0]$, i.e., the first bit of the message after the SOTP, is a -1, while the $G(r)[0] = 1$. This follows because, as we have shown above, adding 2 to a coefficient that is not -1 will cause a reduction modulo 3 to occur, which consequently changes m' and thus r. But, if the first bit of m is not -1, we can target the second bit of m and $G(r)$ by sending $c' = c + 2x$. We continue attempting this idea with $m[i]$ and $G(r)[i]$ for $i \in \{0, 1, \ldots, n-1\}$, until the attack is successful. Since there are four options for these two bits, the success probability is $\frac{1}{4}$, and thus we will have to make four queries on average.

To see why we want that $m[i]$ equal to -1 and $G(r)[i]$ equals 1, we can explore this at the hand of two examples, where we change the first bit of m^+:

Example 11. Let $m^+ = (0, 0, 1, 1)$ and $u_1 \| u_2 = (1, 1, 0, 1, 1, 0, 1, 0)$. Then,

$$\begin{aligned} m &= \mathsf{SOTP}((0,0,1,1), (1,1,0,1), (1,0,1,0)) \\ &= ((0,0,1,1) \oplus (1,1,0,1)) - (1,0,1,0) \\ &= (1,1,1,0) - (1,0,1,0) \\ &= (0,1,0,0) \end{aligned}$$

Then, if an attacker were to create $c' = c + (2, 0, 0, 0)$ by adding $(2, 0, 0, 0)$ to a valid ciphertext c, then

$$\begin{aligned} m' &= f \cdot (c + 2) \in R_q \\ &= f \cdot c + 2 \cdot 3f' + 2 \\ &= m + 2 \mod {}^{\pm}3. \end{aligned}$$

So $m' = (0, 1, 0, 0) + (2, 0, 0, 0) = (2, 1, 0, 0) = (-1, 1, 0, 0) \mod {}^{\pm}3$. Then, the m^{+} is obtained via

$$\begin{aligned} m^{+\prime} &= \mathsf{Inv}((-1, 1, 0, 0), (1, 1, 0, 1), (1, 0, 1, 0)) \\ &= ((-1, 1, 0, 0) + (1, 0, 1, 0)) \oplus (1, 1, 0, 1) \\ &= (0, 1, 1, 0) \oplus (1, 1, 0, 1) \\ &= (1, 0, 1, 1) \\ &\neq m^{+} \end{aligned}$$

and thus, the modulo ${}^{\pm}3$ on m' causes different $m^{+\prime}$ as output, and moreover this changes the r'.

So indeed, if $m[0] \neq -1$ and $G(r)[0] = 1$, we find a different

$$r' = (c' - m') \cdot h^{-1},$$

which causes the decryption to fail regardless. Now, if we take a m^{+} such that $m[0] = -1$ we will find that this does work.

Example 12. Let $m^{+} = (1, 0, 1, 1)$ and $u_1 \| u_2 = (1, 1, 0, 1, 1, 0, 1, 0)$. Then,

$$\begin{aligned} m &= \mathsf{SOTP}((1, 0, 1, 1), (1, 1, 0, 1), (1, 0, 1, 0)) \\ &= ((1, 0, 1, 1) \oplus (1, 1, 0, 1)) - (1, 0, 1, 0) \\ &= (0, 1, 1, 0) - (1, 0, 1, 0) \\ &= (-1, 1, 0, 0) \end{aligned}$$

Then, if an attacker were to create $c' = c + (2, 0, 0, 0)$ by adding $(2, 0, 0, 0)$ to a valid ciphertext c, then

$$\begin{aligned} m' &= f \cdot (c + 2) \mod {}^{\pm}3 \in R_q \\ &= f \cdot c + 2 \cdot 3f' + 2 \\ &= m + 2. \end{aligned}$$

So $m' = (-1, 1, 0, 0) + (2, 0, 0, 0) = (1, 1, 0, 0)$. Then, the m^{+} is obtained via

$$\begin{aligned} m^{+\prime} &= \mathsf{Inv}((1, 1, 0, 0), (1, 1, 0, 1), (1, 0, 1, 0)) \\ &= ((1, 1, 0, 0) + (1, 0, 1, 0)) \oplus (1, 1, 0, 1) \\ &= (2, 1, 1, 0) \oplus (1, 1, 0, 1) \\ &= (0, 1, 1, 0) \oplus (1, 1, 0, 1) \\ &= (1, 0, 1, 1) \\ &= m^{+} \end{aligned}$$

Here, the m' and m are inherently different as both are valid, with only entries in $\{-1, 0, 1\}$. Moreover, the incorrect m' does yield the correct m^{+}, and with that the shared secret key K. Because $r' = r$, as shown in Equation 8, this attack breaks the CCA security of the NTRU+ KEM. To observe why, recall that in the IND-CCA security game from Section 2.3.1, the adversary can pick $c_i' = c_i + 2 \cdot x^j$ which is related but not equal to the ciphertext c_i, and have it decrypted by the decryption oracle. Here, there is a large probability that we can easily find a j such that $m[j] = -1$ and $G(r)[j] = 1$, so the decryption oracle will return the m_i that the challenger has chosen.

A very similar setting would be if $m[0] = 1$, then we can edit the ciphertext c as $c' = c - 2$, which would not trigger a modulo reduction. However, note that when $m[0] = 1$, it requires $u_2[0] = 0$ in the SOTP function. Therefore, when $m'[0] = m[0] - 2 = -1$ is inserted into the Inv function, this will cause

$$(m' + u_2)[0] = -1,$$

and when this is bitwise XORed with $u_1[0]$, we are unsure how the -1 will be interpreted by the computer. Most likely, the input will be taken modulo 2, causing the -1 to be interpreted as a 1, which would make the attack work. However, since we are inserting tampered ciphertexts into the function, it is plausible that

$$\begin{aligned} f \cdot c' &= f \cdot (c - 2) \\ &= 3 \cdot g \cdot r + f \cdot m - 2 \cdot 3f' - 2 \end{aligned}$$

requires a reduction modulo q in the first component. This would change the bit inserted to the XOR to a 0, hence we are not sure what would happen in this case. It is plausible that $m[0] = 1$ is also effective, but for this thesis we will only consider $m[0] = -1$.

To prevent this vulnerability, the NTRU+ team, together with Lee, suggest that a check should be implemented on the $y + u_2$ term in the Inv function [25]. Simply checking that every coefficient of $y + u_2 \in \{0, 1\}$ and outputting a decryption failure otherwise then prevents an edited ciphertext from staying undetected, and yielding the same message m^+.

As a test, to see if this countermeasure can be effective, let us try to edit a ciphertext c by adding any power of x:

$$c' = c + x^k$$

for $k \in \{0, 1, \ldots, n-1\}$. Then the decapsulation would give

$$\begin{aligned} m' &= f \cdot (c + x^k) \mod {}^{\pm}q \\ &= m + (3f' + 1) \cdot x^k \end{aligned}$$

and thus

$$m' = m + x^k \mod {}^{\pm}3.$$

Moreover, it is calculated that

$$\begin{aligned} r &= (c' - m') \cdot h^{-1} \\ &= (c + x^k - (m + x^k)) \cdot h^{-1} \\ &= (c - m) \cdot h^{-1} \end{aligned}$$

and thus r is the same as for an unedited c. Now,

$$\begin{aligned} m' &= \mathsf{Inv}(m', G(r)) \\ &= (m' + u_2) \oplus u_1 \\ &= (m + x^k + u_2) \oplus u_1 \end{aligned}$$

where m is found as $(m^+ \oplus u_1) - u_2$ such that

$$m + x^k + u_2 = (m^+ \oplus u_1) + x^k. \tag{9}$$

Now, with the new check imposed, the decapsulation will output a decryption failure if $(m^+ \oplus u_1) + x^k \notin \{0,1\}^n$. Thus, a decryption failure on $(m^+ \oplus u_1) + x^k$ implies that the k-th component of $(m^+ \oplus u_1) = 1$. But for an attacker both the m^+ and r, from which u_1 is obtained, are secret. Therefore, there are still two options for the k-th component of both m^+ and u_1, and thus we cannot conclude anything about the bits in m^+.

However, this idea can still be useful to create potential attack opportunities. First, recall Section 3.4.6, where we discuss how $(r, K) = H(m^+)$. Thus r is sampled fully from m^+, as the first half of the hash $H(m^+)$. But recall that u_1 is also sampled as the first half of the hash $G(r)$. By abuse of notation, let $G(r)[1,n]$ denote the first n bits of $G(r)$ and similarly, let $H(m^+)[1,n]$ be the first n bits of $H(m^+)$. With this, we can write that

$$m^+ \oplus u_1 = m^+ \oplus G(H(m^+)[1,n])[1,n]. \tag{10}$$

where the left-hand side of the equation is known due to an n-fold reaction attack as shown in Equation 9. Thus, an attacker can brute force different $m^+ \in \{0,1\}^n$ until the the m^+ satisfies Equation 10. Due to the form of m^+, there are only 2^n options, therefore making the search space smaller than for an attack on the private key $f \in \{-1,0,1\}^n$. However, note that in this case, we cannot split the message m^+ into $m_1^+ + m_2^+$, as we did in the brute force attack in Section 4.1, because of the hash function which is applied to m^+.

It should be noted that this attack is not certain to work. First of all, there is some uncertainty on the hash functions used in the C implementation. However, due to our limited understanding of the code we are not able to draw definitive conclusions from the code alone. It is possible that the authors of NTRU+ use a randomized seed in the hash function in order to prevent attacks, like the one above, from happening. If they did not implement this random seed, attacks, such as the one described here, may be able to aid an attacker.

4.4 Quantum Attacks

Although it has been claimed that NTRU, and by extension NTRU+, are post-quantum schemes, we have only seen a single attack which uses a quantum algorithm. This attack, the BKZ lattice reduction as explained in Section 4.2.3, can use Grover's Algorithm to speed up the calculations on a quantum computer.

NTRU and NTRU+ are post-quantum schemes, and thus claim to be resistant against attacks using a quantum algorithm. In the literature there are a few more examples using such algorithms: e.g. May's Meet-LWE attack has a quantum version [39]. Moreover, all the different lattice reduction attacks all have a search phase, which can be replaced to use Grover's algorithm to speed up calculations on a quantum computer. This algorithm, proposed by Grover in 1996, executes an unstructured search leading to a high probability of finding a unique input for a black box function and the corresponding output [16]. Although this algorithm leads to a significant speed up in the search time, and thus a lower security level, it fails to compromise the security of NTRU+.

However, some quantum algorithms, such as Shor's algorithm, will completely break other cryptographic schemes. Therefore, we must wonder whether such an algorithm exists for solving the Shortest Vector Problem (SVP). An analysis of quantum attacks against NTRU was given by Fluhrer in 2015 [13]. He analyses multiple existing attacks, finding that these attacks will be able to break parameter sets with an aim for n bit security in $\frac{n}{2}$ bits. This is the general trend with quantum attacks on NTRU variants: they will significantly lower the required work, but they will not break the system. For example, when aiming for a parameter set with 256 bit security, one should simply use a parameter set targeted at 512 bit security. In practice, Grover speed-up cannot be applied to halve the security level, as it can only be applied to one level of recursion. Therefore, the target of 512 bit security is too ambitious, and in reality less security is often sufficient.

So far, no algorithm has been found which can solve lattice problems, such as the SVP and CVP, in polynomial time. However, that does not mean such an algorithm will not be found in the future. More research is needed to either conclude whether NTRU, and by extension NTRU+, are truly quantum secure.

5 Conclusion

In this thesis, we have discussed the cryptographic scheme NTRU+. We have compared it to previous variants of NTRU systems, in order to see where the similarities and differences are situated. From there, we discussed multiple known attacks on different NTRU variants, while arguing whether or not they can be effective against NTRU+. Now, we will summarize the results obtained from these attacks, and discuss them further. Finally, we will briefly present several attack vectors that we were not able to incorporate, but could offer interesting results nonetheless.

After carefully weighing the effects of different attacks on the NTRU+ KEM, we have not found any vulnerabilities that appear to render the system insecure. The authors have successfully overcome most existing drawbacks of other NTRU schemes, by implementing the Number Theoretic Transform in order to speed up the algorithms used in NTRU. The authors of NTRU+ also claim that their implementation of the ACWC_2 transform and $\overline{\mathrm{FO}}$ transform, build around the SOTP, yields a worst case correctness error. Finally, the security parameters proposed in Chapter 7 of [23] achieve sufficient security levels according to our tests, both for lattice sieving attacks and brute force attacks like Meet-in-the-middle, Golden Collision and Meet-LWE.

One issue with the newly implemented transforms, however, is the message-modification attack on the SOTP as proposed by Lee [25]. This renders the rigidity of the SOTP invalid and thus Lemma 4.3 in [23] does not apply anymore, which leads to a false IND-CCA security proof. Although this seems pretty serious, the attack does not break the scheme, and a simple verification on the intermediate value in the inverse SOTP will fix this issue. This verification may, however, cause even more harm, as we will discuss in Section 5.1.1.

5.1 Future work

There are multiple attacks that we have not yet attempted to implement against NTRU+. Here, we will present a short overview of several attacks that might offer interesting results when applied to NTRU+.

5.1.1 Further reaction attacks against the Inv verification

A first suggestion that allows for future research is the implementation of a standard reaction attack on NTRU+. As described in Section 4.3.1, reaction attacks can be very effective on NTRU variants. However, the NTRU+ CCA-KEM appears to completely defeat this idea. Instead, we focus on the found vulnerability in the SOTP and the potential solution to overcome this vulnerability. In Section 4.3.4, we discussed the problem with the SOTP, found by Lee [25]. Furthermore, it was mentioned that both the authors of NTRU+ and Lee consider a check, which checks whether $m + u_2$ in the Inv function is binary, an adequate solution. However, in Section 4.3.4 we also show how the proposed solution can be abused to lower the search space for the message m. Moreover, NTRU+ was originally intended as a system where decryption failures are not possible. But the attack identified by Lee, and the proposed check change this. Therefore, the error patterns caused by these checks should be researched further.

Another interesting result is the more in-depth analysis of decryption failure attacks on lattice-based schemes by D'Anvers, Guo, Johansson, Nilsson, Vercauteren and Verbauwhede [12]. In their paper, they present a technique called failure boosting, aimed at increasing the amount of decryption failures that occur in a lattice based IND-CCA secure scheme. Moreover, they investigate exactly how much information an adversary will obtain from a single decryption failure. Finally, they propose a reaction attack that can notably lower the security of lattice-based schemes if they have a relatively high failure rate.

5.1.2 Side channel attack on the Number Theoretic Transform

Side channel attacks are attacks which do not directly attack the cryptographic system, but instead analyze and exploit observable features of a computer system. Common examples are to observe the power that a computer system consumes, from which the attacker attempts to infer some information about the data being handled.

Another option is to analyze the time it takes for a system to perform different operations, thus possibly leaking information on the encryption algorithms or data that is being used.

In a thesis from 2022, Custers discusses a Soft Analytical Side-Channel Attack against a Number Theoretic Transform in Post-Quantum Cryptography [9]. Although his thesis focuses on rings of the form

$$R_q[x]/(x^n \pm 1),$$

we think it is possible to adapt their ideas to a ring of the form

$$R_q[x]/(x^n - x^{n/2} + 1).$$

We are not sure how effective these types of attacks will be against the NTRU+ system, but suggest that this is more thoroughly researched in the future.

Another side-channel assisted attack against NTRU was proposed in 2021 by Askeland and Rønjom [3]. Again, it is not certain whether this attack can be adapted against NTRU+ successfully, but we again suggest further investigation into this attack.

5.1.3 Subfield logarithm attack

In a blog post from 2014, Bernstein proposes an attack against ideal lattices, where the ideal can be split up in many smaller polynomials. The idea behind the attack is that we apply the information known about the secret key to a subfield, on which we then run a lattice reduction algorithm to find the secret key. Finally, the found secret key is lifted back to the full field. As was discussed in Section 3.3, the choice of the polynomial $x^n - x^{n/2} + 1$ together with the q cause the NTRU+ quotient polynomial to be split into many different factors.

A similar attack, as analysed by Albrecht, Bai and Ducas, finds subfield weaknesses in NTRU systems where the parameters are "overstretched" [1]. Here, overstretched means that there is a large difference between the size of q and the size of n. Although it appears that the NTRU+ parameter sets are chosen such that this attack does not apply, we still suggest some further research into this attack and its potential target range.

5.1.4 Further risks on lattice-based cryptography

In an extensive paper from 2021, the NTRU Prime risk assessment team writes about the many known risks of lattice-based cryptography [38]. In the paper, the authors discuss attack avenues on lattice-based KEMs which have not yet been ruled out by theorems. With the massive attack surface for lattice-based schemes, it is likely that some seriously dangerous attacks will go unnoticed for some time after the presentation of a new KEM. The paper warns about different attack vectors, many of which appear to apply to NTRU+. Although this thesis considered multiple attacks, this paper offers a more in-depth analysis of the different attack vectors on lattice-based cryptography Therefore, we suggest careful consideration of the methods and risks discussed in this paper.

References

[1] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 153–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

[2] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the LWE, NTRU schemes! In Dario Catalano and Roberto De Prisco, editors, *SCN 18: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 351–367, Amalfi, Italy, September 5–7, 2018. Springer, Heidelberg, Germany.

[3] Amund Askeland and Sondre Rønjom. A side-channel assisted attack on NTRU. Cryptology ePrint Archive, Report 2021/790, 2021. https://eprint.iacr.org/2021/790.

[4] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24, Arlington, VA, USA, January 10–12, 2016. ACM-SIAM.

[5] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. NTRU prime: Reducing attack surface at low cost. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017: 24th Annual International Workshop on Selected Areas in Cryptography*, volume 10719 of *Lecture Notes in Computer Science*, pages 235–260, Ottawa, ON, Canada, August 16–18, 2017. Springer, Heidelberg, Germany.

[6] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. Flush, gauss, and reload - A cache attack on the BLISS lattice-based signature scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 323–345, Santa Barbara, CA, USA, August 17–19, 2016. Springer, Heidelberg, Germany.

[7] Don Coppersmith and Adi Shamir. Lattice attacks on NTRU. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 52–61, Konstanz, Germany, May 11–15, 1997. Springer, Heidelberg, Germany.

[8] Jolijn Cottaar. GoldSIDH: Combining SIDH and the Goldilocks prime. Master's thesis, Eindhoven University of Technology, 2022. https://research.tue.nl/en/studentTheses/goldsidh.

[9] Frank Custers. Soft Analytical Side-Channel Attacks on the Number Theoretic Transform for Post-Quantum Cryptography. Master's thesis, Eindhoven University of Technology, 2022. https://pure.tue.nl/ws/portalfiles/portal/211137599/Custers_F.pdf.

[10] The Sage Developers. SageMath, the Sage Mathematics Software System (Version 9.5), 2022.

[11] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[12] Jan-Pieter D'Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. Decryption failure attacks on IND-CCA secure lattice-based schemes. In *Public-Key Cryptography–PKC 2019: 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II 22*, pages 565–598. Springer, 2019.

[13] Scott Fluhrer. Quantum cryptanalysis of NTRU. Cryptology ePrint Archive, Report 2015/676, 2015. https://eprint.iacr.org/2015/676.

[14] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 26(1):80–101, January 2013.

[15] J.P. Gram. Über die Entwickelung reeller Functionen in Reihen mittelst der Methode der kleinsten Quadrate. *Journal für die reine und angewandte Mathematik*, 1883(94):41–73, 1883.

[16] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery.

[17] Chris Hall, Ian Goldberg, and Bruce Schneier. Reaction attacks against several public-key cryptosystems. In Vijay Varadharajan and Yi Mu, editors, *ICICS 99: 2nd International Conference on Information and Communication Security*, volume 1726 of *Lecture Notes in Computer Science*, pages 2–12, Sydney, Australia, November 9–11, 1999. Springer, Heidelberg, Germany.

[18] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, Heidelberg, Germany, June 1998.

[19] Jeffrey Hoffstein and Joseph H. Silverman. Reaction Attacks Against the NTRU Public Key Cryptosystem, July 2000. https://ntru.org/f/tr/tr015v2.pdf.

[20] Jeffrey Hoffstein and Joseph H. Silverman. Optimizations for NTRU. In *Public-key cryptography and computational number theory. Proceedings of the international conference organized by the Stefan Banach International Mathematical Center, Warsaw, Poland, September 11–15, 2000*, pages 77–88. de Gruyter, 2001.

[21] Nick Howgrave-Graham, Joseph Silverman, and William Whyte. A Meet-In-The-Middle Attack on an NTRU Private Key, July 2003. https://ntru.org/f/tr/tr004v2.pdf.

[22] Thomas W Judson. *Abstract algebra: theory and applications*. PWS Publishing Company, 1994. http://abstract.ups.edu/download.html.

[23] Jonghyun Kim and Jong Hwan Park. NTRU+: Compact construction of NTRU using simple encoding method. *Korean Post-Quantum Competition*, Nov 2022. https://www.kpqc.or.kr/images/pdf/NTRU+.pdf.

[24] Aleksandr Korkine and Yegor Zolotareff. Sur les formes quadratiques positives. *Mathematische Annalen*, 11:242–292, 1877.

[25] Johee Lee. Attack on NTRU+. https://groups.google.com/g/kpqc-bulletin/c/cxnTcy1oJK8, June 2023.

[26] Arjen K. Lenstra, Hendrik W. Lenstra, and László Miklós Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[27] Vadim Lyubashevsky and Gregor Seiler. NTTRU: Truly fast NTRU using NTT. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(3):180–201, 2019. https://tches.iacr.org/index.php/TCHES/article/view/8293.

[28] Alexander May. How to meet ternary LWE keys. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 701–731, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

[29] Peter L. Montgomery. Modular Multiplication without Trial Division. *Mathematics of Computation*, 44(170):519–521, 1985.

[30] Abderrahmane Nitaj. The Mathematics of the NTRU Public Key Cryptosystem. In Addepalli VN Krishna (Eds.), editor, *Emerging Security Solutions Using Public and Private Key Cryptography: Mathematical Concepts*. IGI Global, 2015.

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

[32] Erhard Schmidt. Zur Theorie der linearen und nichtlinearen Integralgleichungen. I. Teil: Entwicklung willkürlicher Funktionen nach Systemen vorgeschriebener. *Mathematische Annalen*, 63:433–476, 1907.

[33] Claus-Peter Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2):201–224, 1987.

[34] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, and Damien Stehlé. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[35] Gregor Seiler. Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography. Cryptology ePrint Archive, Paper 2018/039, 2018. https://eprint.iacr.org/2018/039.

[36] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press.

[37] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 27–47, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

[38] NTRU Prime Risk-Management Team. Risks of lattice KEMs, Oct 2021. https://ntruprime.cr.yp.to/latticerisks-20211031.pdf.

[39] Iggy Van Hoof, Elena Kirshanova, and Alexander May. Quantum key search for ternary LWE. In Jung Hee Cheon and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, pages 117–132, Daejeon, South Korea, July 20–22, 2021. Springer, Heidelberg, Germany.

[40] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, January 1999.

[41] Christine van Vredendaal. Reduced memory meet-in-the-middle attack against the NTRU private key. *LMS Journal of Computation and Mathematics*, 19(A):43–57, 2016.