NTRU over the Eisenstein Integers

Katherine Jarvis

Thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of Master of Science in
Mathematics [1]

Department of Mathematics and Statistics
Faculty of Science
University of Ottawa

[1]The M.Sc. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Abstract

NTRU is a fast public-key cryptosystem that is constructed using polynomial rings with coefficients in $\mathbb{Z}$. We present ETRU, an NTRU-like cryptosystem based on the Eisenstein integers $\mathbb{Z}[\omega]$ where ω is a primitive cube root of unity. We discuss parameter selection and develop a model for the probabilty of decryption failure. We also provide an implementation of ETRU. We use theoretical and experimental data to compare the security and efficiency of ETRU to NTRU with comparable parameter sets and show that ETRU is an improvement over NTRU in terms of security.

Acknowledgements

I would like to thank my supervisor, Monica Nevins, for her enthusiasm and guidance which have been invaluable to me. I would also like to express my gratitude to my family and friends for their continued support and encouragement.

Dedication

Dedicated to my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

NTRU is a public-key cryptosystem that was presented by Hoffstein, Pipher and Silverman in 1998 in [9]. It was further developed and in 2008 an IEEE standard was issued for NTRU [21]. NTRU operates using polynomial rings with coefficients in $\mathbb{Z}$. NTRU is considered to be a lattice based cryptosystem as its security is based on the conjectured hard lattice problem of finding a shortest vector in a lattice.

A drawback to NTRU is that the decryption can sometimes fail, however one can choose parameters so that there is a very small probability of decryption failure. On the other hand, NTRU has multiple advantages. NTRU takes $O(N^2)$ operations to encrypt or decrypt messages of length N which is considerably faster than the $O(N^3)$ operations required by RSA, another public-key cryptosystem [9]. NTRU is also considered to be resistant to quantum computing attacks, unlike other public-key cryptosystems such as RSA and Elliptic Curve Cryptography [15].

To further improve the security of NTRU variations have been proposed using polynomial rings with coefficients in rings other than $\mathbb{Z}$. In 2005, Kouzmenko suggests using other Euclidean rings and presents GNTRU, NTRU over the Gaussian integers [25]. In 2009, Nevins, KarimianPour and Miri extend Kouzmenko's idea and present ETRU, NTRU over the Eisenstein integers [29]. Other variations include MaTRU,

based on matrix rings, presented by Coglinases and Goi in 2005 [3] and QTRU, based on Quaternion algebra, presented by Malekian, Zakerolhosseini and Mashatan in 2009 [27].

In [24] KarimianPour introduces ETRU, NTRU over the Eisenstein integers. Her work is further extended in [29] by Nevins, KarimianPour and Miri where they introduce a division algorithm for the Eisenstein integers and a lattice for ETRU. In this thesis we build on Nevins, KarimianPour and Miri's work. Our goal is to implement ETRU and show that ETRU is an improvement to NTRU in terms of security and efficiency. Our contributions include extending some common attacks against NTRU to ETRU. We also develop a model for the probability of decryption failure for ETRU. We provide an implementation of ETRU and NTRU using C++ and gather experimental data to test the speed and security of ETRU against NTRU. We observe that the lattice security of ETRU seems to be higher than that of NTRU for similar length keys.

The thesis is organized as follows. We begin by presenting some basic properties and notation of lattices in Chapter 2. We also introduce the closest and shortest vector problems and the LLL and BKZ lattice basis reduction algorithms.

In Chapter 3 we describe the NTRU cryptosystem. We discuss parameter selection and some common attacks against NTRU such as the meet-in-the-middle attack and lattice attacks. We develop our own version of the meet-in-the-middle attack in the case where NTRU parameter p is equal to 3, which does not appear in the literature. A theoretical analysis of the encryption and decryption speed of NTRU is provided and we test our analysis with experimental data from our implementation of NTRU. We also describe how to extend NTRU to other rings and give a summary of Kouzmenko's results for GNTRU including his division algorithm for the Gaussian integers. We interpret the Gaussian integers in terms of a lattice and show Kouzmenko's division algorithm is a solution to the closest vector problem.

In Chapter 4 we present properties of Eisenstein integers. We correct the division

algorithm for the Eisenstein integers presented in [29] and go on to implement it. We discuss parameter selection for ETRU and develop a new model for the probability of decryption failure. Our theoretical model is then tested against experimental data gathered from our implementation of ETRU and shown to be accurate.

We discuss the security of ETRU in Chapter 5. We extend the meet-in-the-middle and lattice attacks to ETRU and present our own ETRU lattice. We also discuss the speed of the ETRU encryption and decryption algorithms. We implement lattice attacks against NTRU and ETRU and gather experimental data to compare the speed and security of ETRU to that of NTRU. Our calculations and data appear to suggest that ETRU has stronger lattice security and comparable speed to that of NTRU with similar key lengths.

In Chapter 6 we present our conclusions and discuss possible avenues for future work.

Chapter 2

Lattices and Background

NTRU is a lattice-based cryptosystem and its security is based on the conjectured difficulty of finding short vectors in a certain lattice. In this chapter we give background on lattices and their properties. We also define the shortest and closest vector problems and present the best known algorithms for finding short vectors in a lattice. These are the basis of well known attacks against NTRU, as we will see in Chapter 3.

2.1 Definitions and Notation

We begin by giving some basic definitions and properties of lattices. The material from this section is standard and can be found for example in [10] and [28].

Definition 2.1.1 *Let $\{b_1, \ldots, b_n\} \subset \mathbb{R}^m$ be a set of linearly independent vectors. The* ***lattice*** *generated by $\{b_1, \ldots, b_n\}$ is the set*

$$\mathcal{L}(b_1, \ldots, b_n) = \{a_1 b_1 + \cdots + a_n b_n : a_i \in \mathbb{Z}\}$$

of all integer linear combinations of $b_1, \ldots, b_n$. The integers n and m are respectively called the ***rank*** *and* ***dimension*** *of $\mathcal{L}$. If $n = m$ then $\mathcal{L}$ is called* ***full rank****.*

One can show that a lattice is a *discrete* additive subgroup of $\mathbb{R}^m$ [10]. Remark that later in this thesis we will consider lattices in the complex plane $\mathbb{C}$ by identifying elements of $\mathbb{C}$ with elements of $\mathbb{R}^2$.

A lattice basis can be represented by a $n \times m$ matrix

$$B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

where the basis vectors form the rows of B. We use the notation $\mathcal{L}(B)$ to denote the lattice generated by the rows of matrix B. We will interchangeably use the notation for the matrix B and the set of basis vectors $B = \{b_1, \ldots, b_n\}$.

Note that there will be many different bases for the same lattice. We say that two bases $B, B' \in \mathbb{R}^{n \times m}$ are **equivalent** if $\mathcal{L}(B) = \mathcal{L}(B')$. If two bases B, B' are equivalent then the basis vectors of B' consist of integer linear combinations of the basis vectors of B and vice versa. Recall that a **unimodular** matrix is a square integer matrix with determinant equal to ± 1, so that U^{-1} is also an integer matrix. Thus two bases $B, B' \in \mathbb{R}^{n \times m}$ are equivalent if and only if there exists a unimodular matrix $U \in \mathbb{Z}^{n \times n}$ such that $B' = UB$.

Let $u = (u_1, \ldots, u_m) \in \mathbb{R}^m$ and $v = (v_1, \ldots, v_m) \in \mathbb{R}^m$. Recall that the **scalar product** of u and v is

$$\langle u, v \rangle = \sum_{i=1}^{m} u_i v_i$$

and the **Euclidean norm** of v is

$$\|v\| = \sqrt{\langle v, v \rangle} = \sqrt{\sum_{i=1}^{m} v_i^2}.$$

In later chapters we will also work with complex vectors. Let $w = (w_1, \ldots, w_m) \in \mathbb{C}^m$ where $w_j = a_j + b_j i \in \mathbb{C}$ for $1 \leq j \leq n$. We define

$$\|w\| = \sqrt{\sum_{j=1}^{m} (a_j^2 + b_j^2)}.$$

Definition 2.1.2 *The **fundamental parallelepiped** of $\mathcal{L}(B)$ is*

$$\mathcal{P}(\mathcal{L}(B)) = \{xB : 0 \leq x_i < 1, \forall i = 1, \ldots, n\}.$$

The fundamental parallelepiped does not contain any lattice points other than the origin, and contains a unique representative of each coset of the subgroup $\mathcal{L}(B)$ of $span_{\mathbb{R}}B$.

Definition 2.1.3 *The **determinant** of a lattice $\mathcal{L}(B)$ is defined to be*

$$\det(\mathcal{L}(B)) = \sqrt{\det(BB^T)}.$$

The determinant of a lattice $\mathcal{L}(B)$ is the n-dimensional volume of the fundamental parallelepiped $\mathcal{P}(\mathcal{L}(B))$. Any two equivalent bases have the same determinant, hence the volume of a fundamental parallelepiped is independent of the chosen basis.

Definition 2.1.4 *The **successive minima** associated to an m-dimensional lattice $\mathcal{L}$ of rank n are the constants $\lambda_1, \ldots, \lambda_n$ defined by*

$$\lambda_i(\mathcal{L}) = \inf\{r : \dim(span(\mathcal{L} \cap \mathcal{B}_m(0, r))) \geq i\}$$

where $\mathcal{B}_m(0, r) = \{x \in \mathbb{R}^m : \|x\| < r\}$ is the m-dimensional open ball of radius r centered around 0. That is the ith minimum λ_i is the radius of the smallest closed ball centered about the origin containing i linearly independent vectors of the lattice.

It is clear that we have $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ and that λ_1 is the length of the shortest non-zero vector in the lattice.

2.2 The Shortest and Closest Vector Problems

There are two fundamental computational problems associated to a lattice:

Definition 2.2.1 *The **Shortest Vector Problem** (SVP): Given a lattice $\mathcal{L}$ find a nonzero vector $v \in \mathcal{L}$ such that $\|v\| \leq \|v'\|$ for all nonzero $v' \in \mathcal{L}$.*

Definition 2.2.2 *The **Closest Vector Problem** (CVP): Given a lattice $\mathcal{L}$ and a vector w in $\mathbb{R}^m$, find a vector $v \in \mathcal{L}$ minimizing the distance $\|w - v\|$.*

Note that there may be more than one shortest vector in lattice and similarly more than one closest vector thus we look for *a* shortest vector, not *the* shortest vector.

Both the SVP and CVP are considered difficult. In general, CVP is known to be NP-hard and SVP is NP-hard under a certain randomized reduction hypothesis [10]. The hardness of solving SVP has led to approximate versions of these problems that look for vectors within some factor γ of the optimal solution. We will only consider the approximate version of the SVP:

Definition 2.2.3 ***Approximate Shortest Vector Problem**: Given a lattice $\mathcal{L}$ and a real number $\gamma > 1$, find a nonzero vector $v \in \mathcal{L}$ such that $\|v\| \leq \gamma \|v'\|$ for all nonzero $v' \in \mathcal{L}$.*

In Section 2.3 we will see a deterministic polynomial time algorithm for solving the Approximate SVP for certain γ.

The CVP is, however, easy to solve on a rectangular lattice. For example consider the rectangular lattice in $\mathbb{R}^2$ that is generated by the orthogonal basis $B = \{(a, 0), (0, b)\}$ for some $a, b \in \mathbb{R}$ as illustrated in Figure 2.1. Suppose we want to find the closest vector in $\mathcal{L}(B)$ for some arbitrary $\alpha = (x, y) \in \mathbb{R}^2$. We can round to the nearest horizontal and vertical coordinates by setting

$$\begin{aligned} \gamma &= (a \lfloor x/a \rceil, b \lfloor y/b \rceil) \quad \text{and} \\ \rho &= (x - a \lfloor x/a \rceil, y - b \lfloor y/b \rceil) \end{aligned} \tag{2.2.1}$$

where $\lfloor \rceil$ denotes the nearest integer function. To be precise, $\lfloor a \rceil = \lfloor a \rfloor$ if $|a - \lfloor a \rfloor| < 0.5$ otherwise $\lfloor a \rceil = \lceil a \rceil$. The vector γ is a closest vector to α in $\mathcal{L}(B)$, that is, $\gamma \in \mathcal{L}(B)$ and the expression $\|\rho\| = \|\alpha - \gamma\|$ is minimized. More generally, given an

orthogonal basis B of a lattice in $\mathbb{R}^m$ we would write α in coordinates relative to an orthonormal basis parallel to B and then use a generalization of (2.2.1) to find the closest vector.

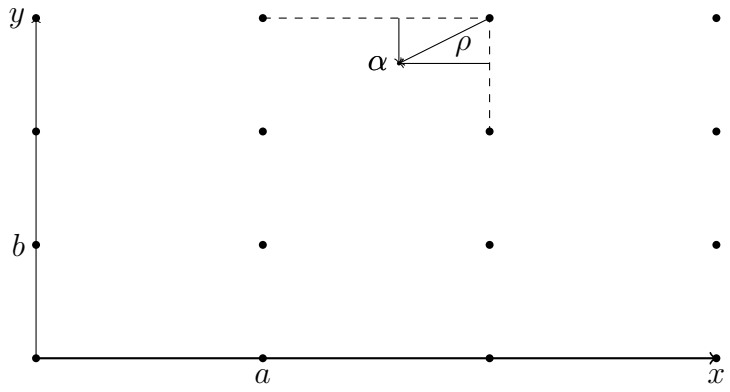

Figure 2.1: We can easily solve the CVP on a lattice in $\mathbb{R}^2$ generated by the basis $B = \{(a, 0), (0, b)\}$ for some $a, b \in \mathbb{R}$ by rounding to the nearest vertical and horizontal coordinates.

The SVP is also easy to solve on a rectangular lattice. Given an orthogonal basis for the lattice, the shortest basis vector is a shortest vector in the lattice. For example suppose $B = \{b_1, b_2, \ldots, b_n\} \subseteq \mathbb{R}^m$ is an orthogonal lattice basis. Then any non-zero $x \in \mathcal{L}(B)$ can be written as an integer linear combination of the basis vectors, that is we can write $x = \sum_{i=1}^{n} a_i b_i$ with $a_i \in \mathbb{Z}$ and $\|x\|^2 = \sum_{i=1}^{n} |a_i|^2 \|b_i\|^2 \geq \min\left\{\|b_i\|^2\right\}$.

2.3 Lattice Basis Reduction

Lattice reduction is a method used for finding short vectors in a lattice. It is based on the observation that if a basis is orthogonal, then it contains a shortest vector. However most lattices do not have an orthogonal basis. Therefore basis reduction looks for a 'nearly' orthogonal basis of the lattice. We look at two lattice reduction algorithms: the LLL algorithm and the BKZ algorithm. The material in this section is drawn from [6] and [28].

2.3.1 The LLL Algorithm

The LLL basis reduction algorithm was presented by Lenstra, Lenstra and Lovasz in 1982 in [26]. The LLL algorithm is a deterministic polynomial time algorithm that finds a solution to the Approximate SVP.

Recall that for any ordered lattice basis $B = \{b_1, b_2, \ldots, b_n\} \subseteq \mathbb{R}^m$ the set of corresponding **Gram-Schmidt** orthogonalized vectors $B^* = \{b_1^*, b_2^*, \ldots, b_n^*\} \subseteq \mathbb{R}^m$ is defined by

$$\begin{aligned} b_1^* &= b_1 \\ b_i^* &= b_i - \sum_{j=1}^{i-1} \mu_{ij} b_j^* \quad \text{for } i = 2, \ldots, n \end{aligned}$$

where

$$\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}.$$

are called the Gram-Schmidt coefficients. Note that b_j^* is orthogonal to b_i for all $i < j$. For the remainder of this chapter B^* will denote the set of Gram-Schmidt orthogonalized vectors corresponding to B.

Note that

$$\begin{aligned} \langle b_i, b_i^* \rangle &= \left\langle b_i^* + \sum_{j<i} \mu_{i,j} b_j^*, b_i^* \right\rangle \\ &= \langle b_i^*, b_i^* \rangle + \sum_{j<i} \mu_{i,j} \langle b_j^*, b_i^* \rangle \end{aligned}$$

and since b_j^* and b_i^* are orthogonal we have

$$\langle b_i, b_i^* \rangle = \langle b_i^*, b_i^* \rangle .$$

It follows that $\mu_{i,i} = 1$.

Since a lattice $\mathcal{L}(B)$ contains only integer linear combinations of the basis vectors in B, B^* is most likely not a basis for $\mathcal{L}(B)$. Recall that $\lfloor a \rceil$ is the nearest integer to a. To attempt to obtain a nearly orthogonal basis of $\mathcal{L}(B)$ we can modify the

Gram-Schmidt algorithm by subtracting only integer multiples of the basis vectors as follows:

$$\begin{aligned} b'_1 &= b_1 \\ b'_i &= b_i - \sum_{j=1}^{i-1} \lfloor \mu_{ij} \rceil \, b'_j \quad \text{for } i = 2, \ldots, n. \end{aligned}$$

Definition 2.3.1 *An ordered basis $B = \{b_1, b_2, \ldots, b_n\} \subseteq \mathbb{R}^m$ of a lattice $\mathcal{L}$ is* ***size reduced*** *if $|\mu_{i,j}| \leq 1/2$ for $1 \leq j < i \leq n$ where the $\mu_{i,j}$'s are the Gram-Schmidt coefficients.*

For example, an ordered basis is size reduced if the basis does not change when we apply the integer version of the Gram-Schmidt algorithm.

Let us define the projection $\pi_i : \mathbb{R}^m \to \text{span}(b_i^*, \ldots, b_n^*)$ by

$$\pi_i(x) = \sum_{j=i}^{n} \frac{\langle x, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} b_j^*. \tag{2.3.1}$$

That is, for any $x \in \text{span}(B)$, $\pi_i(x)$ is the component of x orthogonal to the span of $\{b_1, \ldots, b_{i-1}\}$. Notice that since $\mu_{i,i} = 1$ the Gram-Schmidt vectors can be expressed as $b_i^* = \pi_i(b_i)$.

Definition 2.3.2 *An ordered basis $B = \{b_1, b_2, \ldots, b_n\} \subseteq \mathbb{R}^m$ of a lattice $\mathcal{L}$ is called* ***LLL-reduced*** *with parameter δ (or δLLL-reduced), where δ is a constant satisfying $1/4 < \delta \leq 1$, if it is size-reduced and if for any two consecutive vectors b_i and b_{i+1} we have*

$$\delta \left\| b_i^* \right\|^2 \leq \left\| b_{i+1}^* + \mu_{i+1,i} b_i^* \right\|^2.$$

Note that $b_{i+1}^* + \mu_{i+1,i} b_i^* = \pi_i(b_{i+1})$ so this is equivalent to

$$\delta \left\| b_i^* \right\|^2 \leq \left\| \pi_i(b_{i+1}) \right\|^2.$$

We want to develop an upper bound for the length of the vectors in a δLLL-reduced basis. First we develop a lower bound for λ_1.

Theorem 2.3.3 *[28, Theorem 1.1] Let B be a lattice basis and let B^* be the corresponding set of Gram-Schmidt orthogonalized vectors. Then, the first minimum of the lattice $\mathcal{L}(B)$ satisfies*

$$\lambda_1 \geq \min_j \left\|b_j^*\right\|.$$

Proof: Let us consider an arbitrary lattice vector xB for some nonzero $x \in \mathbb{Z}^m$. It is clear that $\lambda_1 = \inf \|xB\|$. We let i be the biggest index such that $x_i \neq 0$. We will show that $\|xB\| \geq \|b_i^*\| \geq \min_j \left\|b_j^*\right\|$ and therefore it follows that $\lambda_1 \geq \min_j \left\|b_j^*\right\|$.

We know that $xB = \sum_{j=1}^{i} b_j x_j$. Hence

$$\langle xB, b_i^* \rangle = \sum_{j=1}^{i} \langle b_j, b_i^* \rangle \, x_j.$$

Since b_i^* is orthogonal to $b_1, \ldots, b_{i-1}$ and $\langle b_i, b_i^* \rangle = \langle b_i^*, b_i^* \rangle = \|b_i^*\|^2$ we have

$$\begin{aligned} \langle xB, b_i^* \rangle &= \langle b_i, b_i^* \rangle \, x_i \\ &= \|b_i^*\|^2 \, x_i. \end{aligned}$$

Since $|x_i| \geq 1$ we have

$$|\langle xB, b_i^* \rangle| \geq \|b_i^*\|^2.$$

Recall the Cauchy-Schwarz inequality that $|\langle xB, b_i^* \rangle| \leq \|xB\| \, \|b_i^*\|$ hence $\|xB\| \, \|b_i^*\| \geq \|b_i^*\|^2$. Since $\|b_i^*\| \neq 0$ it follows that $\|xB\| \geq \|b_i^*\|$. ∎

The following lemma gives an upper bound for the length of the first vector in a LLL-reduced basis. This is [28, Lemma 2.8], however the proof is my own.

Lemma 2.3.4 *If $B = [b_1 \ldots b_n]^T \in \mathbb{R}^{n \times m}$ is a δLLL-reduced basis with $1/4 < \delta \leq 1$, then*

$$\|b_1\| \leq (2/\sqrt{4\delta - 1})^{n-1} \lambda_1$$

Proof: By the definition of an LLL reduced basis we have that

$$\delta \left\| b_i^* \right\|^2 \leq \left\| b_{i+1}^* + \mu_{i+1,i} b_i^* \right\|^2$$

for all $1 \leq i < n$. Since b_i^* and b_{i+1}^* are orthogonal we have

$$\delta \left\| b_i^* \right\|^2 \leq \left\| b_{i+1}^* \right\|^2 + \mu_{i+1,i}^2 \left\| b_i^* \right\|^2$$

and since $|\mu_{i,j}| \leq 1/2$ for all $i > j$ we have that

$$\delta \left\| b_i^* \right\|^2 \leq \left\| b_{i+1}^* \right\|^2 + \frac{1}{4} \left\| b_i^* \right\|^2 .$$

We can rearrange the terms to get

$$(\delta - \frac{1}{4}) \left\| b_i^* \right\|^2 \leq \left\| b_{i+1}^* \right\|^2 .$$

Next we show by induction on $i - j$ that

$$(\delta - \frac{1}{4})^{i-j} \left\| b_j^* \right\|^2 \leq \left\| b_i^* \right\|^2 \tag{2.3.2}$$

for all $i \geq j$. It is clear that the expression (2.3.2) holds when $j = i$ and $j = i - 1$. We will now assume that the expression (2.3.2) holds for all $k \leq j \leq i$ and show it also holds for $j = k - 1$. Since $(\delta - \frac{1}{4}) > 0$ we have:

$$(\delta - \frac{1}{4}) \left\| b_{k-1}^* \right\|^2 \leq \left\| b_k^* \right\|^2 \leq (\delta - \frac{1}{4})^{-i+k} \left\| b_i^* \right\|^2$$

it follows that

$$(\delta - \frac{1}{4})^{i-(k-1)} \left\| b_{k-1}^* \right\|^2 \leq \left\| b_i^* \right\|^2 .$$

Therefore the above expression holds for all $i \geq j$.

Hence in particular

$$(\delta - \frac{1}{4})^{i-1} \left\| b_1^* \right\|^2 \leq \left\| b_i^* \right\|^2 .$$

Since $b_1^* = b_1$ and $(\delta - \frac{1}{4}) < 1$ we have

$$(\delta - \frac{1}{4})^{n-1} \left\| b_1 \right\|^2 \leq \left\| b_i^* \right\|^2 ,$$

and since by Theorem 2.3.3 $\lambda_1 \geq \min_i \|b_i^*\|$ we have

$$(\delta - \frac{1}{4})^{n-1} \|b_1\|^2 \leq \lambda_1^2.$$

Therefore $\|b_1\| \leq (\delta - \frac{1}{4})^{\frac{-(n-1)}{2}} \lambda_1 = (2/\sqrt{4\delta - 1})^{n-1} \lambda_1$. ∎

Consequently an LLL-reduced basis will give us a lattice vector whose norm is within a factor of $(2/\sqrt{4\delta - 1})^{n-1}$ of the minimum. Notice that for $\delta = 0.99$ an LLL-reduced basis will give us a vector within a factor of approximately 1.16^{n-1} of the minimum. Hence a δLLL-reduced basis will be more likely to contain shorter vectors for larger δ and smaller n.

The **LLL algorithm** takes an ordered basis $B = \{b_1, b_2, \ldots, b_n\}$ of a lattice in $\mathbb{R}^m$ and parameter δ and outputs a *δLLL-reduced* basis consisting of relatively short vectors. Essentially the LLL algorithm consists the following three steps as described in [28]. Set $i = 2$:

1. **Reduction Step:** Size reduce the vector b_i, that is set

$$b_i' = b_i - \sum_{j=1}^{i-1} \lfloor \mu_{i,j} \rceil \, b_j'$$

   where $\lfloor \mu_{i,j} \rceil$ denotes the nearest integer to $\mu_{i,j}$ and replace b_i with b_i'. Notice that the orthogonalized Gram-Schmidt vectors B^* corresponding to B before and after this step are the same. We only have to update the Gram-Schmidt coefficients $\mu_{i,j}$ for $1 \leq j \leq n$.

2. **The Swap Step:** If the consecutive vectors b_i and b_{i-1} do not satisfy the condition for a δLLL reduced basis $\delta \left\|b_{i-1}^*\right\|^2 \leq \left\|b_i^* + \mu_{i,i-1} b_{i-1}^*\right\|^2$, then swap them and set $i = \max(i - 1, 2)$ and update the corresponding Gram-Schmidt basis and coefficients. Otherwise increase i by one.

3. **Repeat:** Repeat from Step 1. The algorithm terminates when $i = n$. Note size reducing a vector b_k does not affect the size-reduction of other vectors, hence when the algorithm terminates we have a size reduced basis. Since the LLL condition holds for all i in Step 2 we have a δLLL reduced basis.

It is clear that the final basis B output by the LLL algorithm is equivalent to the original basis that was input into the algorithm as it has been obtained through invertible elementary integer row operations.

The efficiency of the LLL algorithm has been improved upon since it was originally presented in [26] with $\delta = 3/4$. Efficient pseudocode for the LLL algorithm can be found in [6]. The algorithm presented in [6] terminates after at most $O(m^2 \log B)$ iterations for $\delta < 1$, where $B = \max(\|b_1\|^2, \ldots, \|b_n\|^2)$. Proving that the LLL algorithm terminates in polynomial time for $\delta = 1$ is a long-standing open problem [28, Chapter 2, Section 4].

Shoup's C++ NTL library [32] contains an efficient implementation of the LLL algorithm for a basis with vectors in $\mathbb{Z}^m$. The algorithm in Shoup's C++ NTL library is based on the pseudocode presented by Euchner and Schnorr in [6].

Recall Lemma 2.3.4 shows that the LLL algorithm is only guaranteed to find a vector whose norm is within a certain factor of λ_1. The algorithm in practice finds much shorter vectors than is guaranteed by the worst case bound [6]. The algorithm often finds a shortest vector for lattices of small rank n. However as n increases, the quality of the basis found by the LLL algorithm decreases, that is, the LLL algorithm is less successful in finding short vectors. Therefore we want to look at another algorithm that outputs a higher quality basis.

2.3.2 The BKZ Algorithm

The Block Korkine-Zolotarev (BKZ) reduction algorithm that was introduced by Schnorr in 1987 in [31] combines Korkine-Zolotarev reduction with the LLL algorithm.

Definition 2.3.5 *An ordered basis $B = \{b_1, \ldots, b_n\} \subset \mathbb{R}^m$ of a lattice $\mathcal{L}$ is **Korkine-Zolotarev** (or KZ) reduced if it is size reduced and if*

$$\|b_i^*\| = \lambda_1(\pi_i(\mathcal{L}(B))) \quad \text{for } i = 1, \ldots, n$$

where π_i is the orthogonal projection of $\mathcal{L}$ to span$(b_1, \ldots, b_{i-1})^\perp$ as defined by Equation (2.3.1).

Notice that if we have a KZ reduced basis then $b_1 = b_1^*$ is the shortest vector in the lattice. This is a very strong condition and the fastest known algorithm to find a KZ-reduced basis has exponential running time [6]. Schnorr introduces the notion of a block Korkine-Zolotarev reduced basis in [31]. We define the **blocksize** to be an integer β with $2 \leq \beta < n$.

Definition 2.3.6 *An ordered lattice basis $B = \{b_1, \ldots, b_n\} \subset \mathbb{R}^m$ is β-reduced with parameter δ if it is size reduced and if*

$$\delta \|b_i^*\|^2 \leq \lambda_1(\pi_i \mathcal{L}(b_1, \ldots, b_{\min(i+\beta-1,n)}))^2, \quad \text{for } i = 1, \ldots, n-1$$

with $1/4 < \delta \leq 1$.

Notice that for the special case where $\beta = 2$ we have $\delta \|b_i^*\|^2 \leq \lambda_1(\pi_i \mathcal{L}(b_1, \ldots, b_{i+1}))^2$. This means that

$$\delta \|b_i^*\|^2 \leq \|\pi_i(v_i b_i + v_{i+1} b_{i+1})\|^2$$

for all $v_i, v_{i+1} \in \mathbb{Z}$ with v_i, v_{i+1} not both 0. More specifically the statement must hold in the case where $v_k = 0$ and $v_{k+1} = 1$. Hence

$$\delta \|b_i^*\|^2 \leq \|\pi_i(b_{i+1})\|^2$$

and the basis is δLLL-reduced.

The **BKZ algorithm** takes an ordered basis $B = \{b_1, \ldots, b_n\}$ of a lattice in $\mathbb{R}^m$, blocksize β and parameter δ and outputs a β-reduced basis with parameter δ. Essentially the BKZ algorithm consists of the following three steps. Set $j = 1$:

1. **Reduction Step:** Perform δLLL-reduction on basis B. Note after this step the basis is size reduced.

2. **Block KZ Reduction Step:** Search for a shortest vector b_j^{new} in $\pi_j\mathcal{L}(b_j, \ldots, b_k)$ where $k = \min(j + \beta - 1, n)$. If $\delta \left\|b_j^*\right\|^2 > \left\|b_j^{new}\right\|^2$, that is, the basis is not β-reduced, then δLLL-reduce $\left\{b_1, \ldots, b_{j-1}, b_j^{new}, b_j, \ldots, b_{k+1}\right\}$. Note that as this is not a linearly independent set, the LLL algorithm will return a basis with a zero vector which we remove. Otherwise $\delta \left\|b_j^*\right\|^2 \leq \left\|b_j^{new}\right\|^2$ and the β-reduced condition holds.

3. **Repeat:** Increase j by one (or set $j = 1$ if $j = n - 1$) and repeat step 2 until the β-reduced condition holds for all $j = 1, \ldots, n - 1$.

The BKZ algorithm with $\beta = 2$ is essentially the LLL algorithm which terminates in polynomial time. In general, the BKZ algorithm does not terminate in polynomial time for all β. As the size as the blocksize β increases, the quality of basis improves, in other words shorter vectors are found [28]. However, as β increases the running time of the BKZ algorithm also increases going from polynomial running time to an exponential running time [28].

Pseudocode for the BKZ algorithm can be found in [6]. Shoup's C++ NTL library [32] also contains an efficient implementation for the BKZ algorithm for a basis with vectors in $\mathbb{Z}^m$.

In general basis reduction algorithms such as LLL and BKZ are more successful at finding a shortest vector in a lattice if the length of a shortest vector is much shorter than the shortest expected vector in the lattice [15]. The Gaussian heuristic [30] estimates the shortest non-zero vector in a lattice $\mathcal{L}$ to be

$$s = \sqrt{\frac{d}{2\pi e}}(\det \mathcal{L})^{1/d} \tag{2.3.3}$$

where d is the dimension of the lattice. We will return to this concept in Section 3.3.1.

Chapter 3

NTRU Cryptosystem

In this chapter we describe the original NTRU cryptosystem as presented by Hoffstein, Pipher and Silverman published in 1998 in [9]. We also look at choosing NTRU parameters sets and various attacks against NTRU. We then present NTRU-like cryptosystems. In Section 3.4.1 we discuss NTRU over the Gaussian integers (GNTRU) that was presented by Kouzmenko in 2005 [25]. We emphasize those features we will generalize in Chapter 4 where we extend NTRU over the Eisenstein integers.

3.1 The Cryptosystem

3.1.1 Notation

The NTRU public key cryptosystem as described in [9] depends on three integer parameters N, p and q, such that $N > 1$, p and q are relatively prime and q is much larger that p. We set

$$\mathcal{R} = \mathbb{Z}[x]/\langle x^N - 1\rangle$$

which is the set of convolution polynomials of degree $N-1$. We denote by $*$ multiplication in the ring $\mathcal{R}$. Let $\mathcal{L}_m$, $\mathcal{L}_f$, $\mathcal{L}_g$ and $\mathcal{L}_\phi$ be subsets of $\mathcal{R}$.

We may identify a polynomial in $\mathcal{R}$ with a vector in $\mathbb{Z}^N$. For example we may refer to the polynomial $f = f_0 + f_1 x + \cdots + f_{N-1}x^{N-1}$ as the vector $f = (f_0, f_1, \ldots, f_{N-1}) \in \mathbb{Z}^N$.

Let p be a positive integer. We say that a polynomial $f \in \mathcal{R}$ is **reduced modulo** p if the coefficients of f all lie in the interval $(-\frac{p}{2}, \frac{p}{2}]$. The reason for this choice of interval will be discussed in Section (3.2.5). Given $r \in R$, recall that the notation

$$x \equiv r \mod p$$

means that x is congruent to r modulo p. On the other hand, when we write

$$x = r \mod p$$

we mean to assign to x the unique element of $\mathcal{R}$ with is congruent to r modulo p and is reduced modulo p.

3.1.2 Key Generation

To generate the NTRU keys, we first select two polynomials $f \in \mathcal{L}_f$ and $g \in \mathcal{L}_g$. We take $F_p, F_q \in \mathcal{R}$ satisfying the following properties:

$$F_p * f \equiv 1 \mod p \quad \text{and} \quad F_q * f \equiv 1 \mod q.$$

Note that we need f to be invertible modulo p and q. Next set

$$h = F_q * g \mod q.$$

Notice that this is equivalent to the condition $f * h \equiv g \mod q$.

Our public key is the polynomial h. The parameters N, p and q are also public. Our private key is the polynomial f. For efficiency we may also want to store F_p.

3.1.3 Encryption

Suppose we want to encrypt a message $m \in \mathcal{L}_m$ with coefficients reduced modulo p. First choose a random polynomial $\phi \in \mathcal{L}_\phi$. Our encrypted message is then

$$e = p\phi * h + m \mod q.$$

3.1.4 Decryption

To decrypt an encrypted message e we must first compute

$$a = f * e \mod q.$$

We recover the original message by computing

$$m' = F_p * a \mod p.$$

3.1.5 When Does Decryption Work?

We want to verify that $m' = m$, that is we have recovered our original message m. First we must compute the polynomial a:

$$\begin{aligned} a &= f * e \mod q \\ &\equiv pf * \phi * h + f * m \mod q. \end{aligned}$$

Recall that $f * h \equiv g \mod q$ hence

$$a \equiv p\phi * g + f * m \mod q.$$

Now suppose that the polynomial $p\phi * g + f * m$ has all of its coefficients lying in the interval from $(\frac{-q}{2}, \frac{q}{2}]$. This means that there is no change when we reduce the polynomial modulo q. That is the polynomial a is exactly $p\phi * g + f * m$. In this case when we compute $F_p * a$ we have

$$F_p * a \equiv F_p * (p\phi * g + f * m) \mod p$$

$$\begin{aligned} &\equiv pF_p * \phi * g + m \mod p \\ &\equiv m \mod p \end{aligned}$$

therefore we have recovered the original message m. This shows that under the assumption that the polynomial $p\phi * g + f * m$ is reduced modulo q we have $m = m' = F_p * a \mod p$.

If the coefficients of the polynomial $p\phi * g + f * m$ do not all lie in the interval $(\frac{-q}{2}, \frac{q}{2}]$ then the decryption process will not recover the original message m. In this case we have a **decryption failure**. Note that the recipient would not know that a decryption failure has occurred. However, for appropriate parameter choices we are able to ensure that the polynomial $p\phi * g + f * m$ (almost always) has all of its coefficients in the interval $(\frac{-q}{2}, \frac{q}{2}]$.

3.2 Implementing NTRU

3.2.1 Selecting p and q

The parameters p and q must be relatively prime and q must be significantly larger than p since in the decryption process we need there to be no change when we reduce the coefficients of the polynomial $p\phi * g + f * m$ modulo q. We also want to choose these parameters so that we are able to find inverses of f modulo p and q.

In general, recall that for f to have an inverse in a ring $K = A[x]/\left\langle x^N - 1 \right\rangle$ where A is a field, we need that $\gcd(f, x^N - 1) = 1$. In this case the extended Euclidean algorithm finds an $s, t \in A[x]$ such that $sf + t(x^N - 1) = \gcd(f, x^N - 1) = 1$. It follows that s is the inverse of f in the ring K.

Now consider the problem of finding an inverse of f modulo p. That is we want to find the inverse of f in $\mathcal{R}/\langle p \rangle$. Assume that p is prime. Then $\mathcal{R}/\langle p \rangle$ is isomorphic to $A[x]/\left\langle x^N - 1 \right\rangle$ where $A = \mathbb{Z}/p\mathbb{Z}$. Since $\mathbb{Z}/p\mathbb{Z}$ is a field the extended Euclidean

algorithm can be used to determine whether there exists an inverse of f in $\mathcal{R}/\langle p \rangle$ and will find the inverse if it exists.

If p is a power of a prime, say $p = Q^r$ with Q prime, then there is an algorithm to find the inverse modulo $p = Q^r$ assuming inverse modulo Q is known (see [33]). Hence p and q are often chosen to be primes or prime powers.

A couple of common combinations for p and q are the binary case where $p = 2$ and q is prime or the trinary case where $p = 3$ and $q = 2^n$ [15]. Taking q to be a power of 2 allows for efficient reductions modulo q [8]. Current NTRU parameter often take $q = 2048$ [21]. Using $p = 3$ implies that messages are trinary polynomials with coefficients in $\{-1, 0, 1\}$.

Note that in general the NTRU encryption and decryption algorithms do not require p be an integer. We may take p to be a small polynomial as long as p and q are relatively prime in the ring $\mathcal{R}$. For example current NTRU implementations often take $p = 2 + x$ [21]. Since this is relatively prime to 2, they take q as a power of 2 as in the trinary case and their messages are binary polynomials with coefficients 0 and 1. They also use a different algorithm for reducing a polynomial modulo $p = 2 + x$ as described in [12], it is no longer simply a reduction of coefficients. However, in this thesis we will only be considering the cases were $p = 2$ or $p = 3$.

3.2.2 Selecting N

The parameter N is usually chosen to be prime [8]. It is shown in [7] that there is an attack that can significantly reduce the lattice security of NTRU when choosing N to be composite. Choosing N to be prime also increases the probability that f has an inverse modulo p and q [11].

3.2.3 The Forms of m, f, g, ϕ

Recall that $\mathcal{L}_m$, $\mathcal{L}_f$, $\mathcal{L}_g$ and $\mathcal{L}_\phi$ are subsets of $\mathcal{R}$. Hoffstein, Pipher and Silverman [9] define three new integer parameters d_f, d_g and d_ϕ which determine the number of non-zero entries in f, g and ϕ.

The message space $\mathcal{L}_m$ is chosen such that the coefficients are already reduced modulo p.

In the trinary case ($p = 3$) f, g and ϕ all have coefficients equal to 1, -1 or 0. Motivated, we presume, by the desire to have the coefficients of $a = p\phi * g + f * m$ to be as small as possible, Hoffstein, Pipher and Silverman define $\mathcal{L}_f$, $\mathcal{L}_g$ and $\mathcal{L}_\phi$ as follows. They define $\mathcal{L}_g$ to be the set of polynomials with d_g coefficients equal to 1, d_g coefficients equal to -1 and the rest equal to 0. Similarly $\mathcal{L}_\phi$ is the set of polynomials with d_ϕ 1's, d_ϕ -1's and the rest 0. Notice that since we need f to be invertible we need f to be relatively prime to $x^N - 1$. Since $x - 1$ is clearly a factor of $x^N - 1$ we need to choose f so that $f(1) \neq 0$ which means $x - 1$ is not a factor of f. This means f cannot have an equal number of 1's and -1's. Therefore they choose $\mathcal{L}_f$ to have $(d_f + 1)$ 1's, d_f -1's and the rest 0. Note however, choosing d_f in this fashion does not guarantee that f will be invertible. In the trinary case d_f, d_g and d_ϕ are at most $\lfloor N/3 \rfloor$.

In the binary case ($p = 2$) f, g and ϕ all have coefficients equal to 1 or 0. We define $\mathcal{L}_f$ to be the set of polynomials with d_f coefficients equal to 1 and the rest equal to 0's. Similarly $\mathcal{L}_g$ is the set of polynomials with d_g 1's and $\mathcal{L}_\phi$ is the set of polynomials with d_ϕ 1's. Again we need f to be invertible so we need to choose d_f to be odd.

Another choice as suggested in [12] is to take $f = 1 + p * F$ where F is a binary or trinary polynomial with d_f 1's (and -1's). In this case the inverse of f modulo p is 1, hence key creation and the decryption process are sped up. Current NTRU parameter sets often chosen f in this fashion [15]. However in this paper we will

restrict our attention to the cases where f is a binary or trinary polynomial.

3.2.4 Implementation

We implemented the NTRU encryption and decryption algorithms as described in Section 3.1 NTRU using C++. As we want to be able to compare NTRU to ETRU in Chapters 4 and 5 we developed our own functions for multiplication, division and finding inverses of polynomials in the ring $\mathcal{R}$ instead of using those available in Shoup's NTL C++ library. In Chapter 5 we also implement lattice attacks on the NTRU lattice described in Section 3.3.4.

We use long division to divide polynomials and the extended Euclidean algorithm to find inverses of polynomials modulo a prime integer. To find inverses of polynomials modulo a prime power we use the algorithm described in [33] based on Newton iterations which allows us to compute the inverse modulo prime powers p^r given the inverse modulo the prime p.

All tests are run on a computer with a 2.33GHz Intel Core2 Quad Q8200 processor using the Window Vista(64-bit) operating system.

3.2.5 Decryption Failure

Recall that if the coefficients of the polynomial $\mathrm{A} = p\phi * g + f * m$ in the decryption process do not all lie in the interval $(\frac{-q}{2}, \frac{q}{2}]$ then we will not recover the original message m. In this case we have $\mathrm{A} \neq a \equiv p\phi * g + f * m \mod q$. We write the polynomial A as $\sum \mathrm{A}_i x^i$. In Section 4.4 of [25], Kouzmenko develops a theoretical model for decryption failure by finding the probability that all the coefficients of $\mathrm{A} = p\phi * g + f * m$ lie in the interval $(\frac{-q}{2}, \frac{q}{2}]$ for trinary keys f and g. We will present a similar derivation for the decryption failure of ETRU in Section 4.4.4 therefore we do not include his derivation in this section.

To simplify his presentation Kouzmenko assumes that the polynomial f has d_f

1's and -1's. Kouzmenko calculates that the expected value of each coefficient A_i is $E(\mathrm{A}_i) = 0$ and the variance is

$$Var(\mathrm{A}_i) = \frac{4p^2 d_\phi d_g}{N} + \frac{d_f(p-1)(p+1)}{6}.$$

Assuming the coefficients are normally distributed, the probability that the coefficient A_i lies in $[\frac{-q}{2}, \frac{q}{2}]$ is

$$Prob\left(|\mathrm{A}_i| \leq \frac{q-1}{2}\right) = 2\phi\left(\frac{q-1}{q\sigma}\right) - 1,$$

where $\sigma = \sqrt{Var(\mathrm{A}_i)}$ and ϕ denotes the standard normal distribution. Hence, assuming all the coefficients are independent, the probability that all of the coefficients in A lie in $[\frac{-q}{2}, \frac{q}{2}]$ is

$$Prob\left(\|\mathrm{A}\|_\infty \leq \frac{q-1}{2}\right) = 2\left(\phi\left(\frac{q-1}{q\sigma}\right) - 1\right)^N \tag{3.2.1}$$

where $\|\mathrm{A}\|_\infty = \max\{|\mathrm{A}_0|, \ldots, |\mathrm{A}_{N-1}|\}$. Note that as q increases the probability of decryption failure decreases and as d_f, d_g and d_ϕ increase the probability of decryption failure also decreases.

3.2.6 Encryption and Decryption Speed

In his thesis [25], Kouzmenko develops a formula for the number of elementary operations for the NTRU encryption and decryption algorithms.

To encrypt a message we compute

$$e \equiv p\phi * h + m \mod q.$$

This involves $N^2 + N$ integer multiplications, N integer additions and N integer reductions (divisions) modulo q. Since addition is much faster than multiplication and division we will not count the integer additions. We ran experiments to compare the speed of integer multiplication to integer reduction. We found that it took on

average 0.228 seconds to multiply 100 million sets of two random integers and it took on average 0.231 seconds to preform a 100 million reductions between two random integers. Therefore we take a multiplication to be approximately one division and there are

$$N^2 + 2N \tag{3.2.2}$$

integer multiplications and divisions involved in the NTRU encryption algorithm.

				Time (s)		Formula		Ratio	
N	d_f	d_g	d_ϕ	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt
73	24	24	24	8.30	17.15	5475	10804	659.69	629.82
83	27	27	27	10.59	22.12	7055	13944	666.06	630.25
91	30	30	30	12.72	26.55	8463	16744	665.15	630.78
107	35	35	35	17.60	36.75	11663	23112	662.78	628.84
119	39	39	39	21.75	45.51	14399	28560	662.13	627.62
150	50	50	50	34.41	72.43	22800	45300	662.52	625.40
200	66	66	66	60.95	128.18	40400	80400	662.84	627.23
250	83	83	83	95.34	200.56	63000	125500	660.78	625.76
300	100	100	100	137.69	288.84	90600	180600	661.74	625.26
350	116	116	116	186.98	392.86	123200	245700	658.90	625.42
400	133	133	133	244.66	513.36	160800	320800	657.236	624.89
						Average:		661.80	627.39

Table 3.1: NTRU Encryption and Decryption Speed with $p = 3$ and $q = 2048$. We record the time it takes to encrypt and decrypt 10000 messages. We record the ratio of the number of elementary operations for encryption and decryption to the time it took for encryption and decryption of the 10000 messages.

To decrypt a message first compute

$$a \equiv f * e \mod q$$

which involves N^2 integer multiplication and N reductions modulo q. Next we compute

$$m \equiv F_p * a \mod p$$

which involves another N^2 integer multiplication and N reductions modulo p. Hence there are a total of

$$2N^2 + 2N \tag{3.2.3}$$

integer multiplications and divisions involved in the NTRU decryption algorithm.

We test these formulas by recording the time it takes to encrypt and decrypt 10000 messages in Table 3.1. We then compare the ratio of the number operations given by equations 3.2.2 and 3.2.3 to the time it took to encrypt and decrypt the 10000 messages. Notice that encryption seems to be twice as fast as decryption.

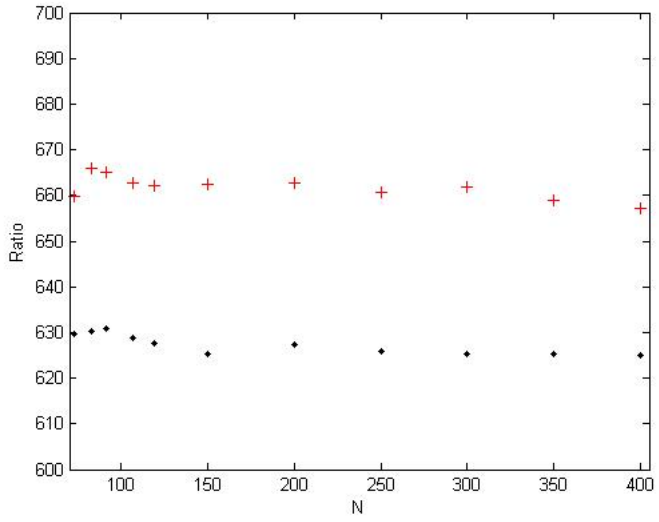

Figure 3.1: Plot of the ratios in Table 3.1. The crosses represent encryption and the dots represent decryption.

The ratios are fairly consistent as depicted in Figure 3.1. There is, however, a gap between the ratios of encryption and decryption. This is likely due to our implementation, similar issues arise and are discussed in Section 5.5.2. Equations (3.2.2) and (3.2.3) seem to be a good representation for comparing the encryption and decryption speed of NTRU. We return to this table in Section 4.4.5 where we compare the speed of ETRU to NTRU.

Note that there are faster methods to multiply truncated polynomials as seen in [34]. Therefore, in practice the NTRU encryption and decryption algorithms can be implemented to be faster than described here.

3.3 Security Analysis

We now look at the main attacks against NTRU and the security of NTRU against these attacks. One method to attack the NTRU cryptosystem is to find the NTRU private key f. One can also attack the message by trying to recover ϕ. In this section we look at attacks to recover f, however these attacks can all be modified to recover ϕ.

3.3.1 Alternate Private Keys

First we make a note about alternate keys f' that can be used to decrypt the same messages as f. Let f' be any rotation of the NTRU private key f, that is let $f' = x^i * f$ for some positive integer i. Since $g \equiv f * h \mod q$ we have that $x^i g \equiv x^i f * h \mod q$. We can see that the polynomial f' can decrypt the same messages as f. Let $g' = x^i * g$ be the corresponding rotation of g. Note that since $g \equiv f * h \mod q$ we also have $g' \equiv f' * h \mod q$.

Suppose we want to decrypt e. First we calculate

$$a' = f' * e = p\phi * g' + f' * m.$$

Note that $a' = x^i(p\phi * g + f * m) = x^i a$ hence if all the coefficients of a lie in the interval $(\frac{-q}{2}, \frac{q}{2}]$ then so do the coefficients of a'. Next we calculate

$$F'_p * a' \equiv pF'_p * \phi * g' + F'_p * f' * m \equiv m \mod p$$

where F'_p is the inverse of f' modulo p. Notice we have recovered our message m. Therefore f' can decrypt the same messages as the NTRU private key f. Similarly

any rotation of $-f$ can also decrypt the same messages as f.

Actually any f', g' pair that satisfies the conditions that $g' \equiv f' * h \mod q$ and f' has and inverse modulo p, can be used to decrypt messages if the coefficients of $a' = p\phi * g' + f' * m$ all lie in the interval $(\frac{-q}{2}, \frac{q}{2}]$. For instance this could happen when the coefficients of f' and g' are small. Such pairs (f', g') are called alternate keys.

3.3.2 Brute Force Attack

One way an attacker can recover the private key, or an alternate key, is to try all possible $f' \in \mathcal{L}_f$. Since $f * h \equiv g \mod q$ we test to see if $f' * h \mod q$ has small (binary or trinary) entries. Similarly an attacker could also try all possible $g' \in \mathcal{L}_g$ and test to see if $g' * h^{-1} \mod q$ has small entries. Therefore the key security depends on the number of elements in $\mathcal{L}_g$ (or $\mathcal{L}_f$, whichever is smaller).

In the binary case ($p = 2$) the size of the key space $\mathcal{L}_g$ is

$$|\mathcal{L}_g| = \binom{N}{d_g} = \frac{N!}{(N - d_g)!d_g!}. \tag{3.3.1}$$

In the trinary case ($p = 3$) the size of the key space $\mathcal{L}_g$ is

$$|\mathcal{L}_g| = \binom{N}{d_g}\binom{N - d_g}{d_g} = \frac{N!}{(N - 2d_g)!(d_g!)^2}. \tag{3.3.2}$$

The key space size is an upper bound on the search time. We know that every rotation of the NTRU key pair (f, g) can be used to decrypt messages. There exists a rotation g' of g where the constant term is 1. Therefore we can slightly reduce the search time by fixing the constant term to be 1 and search for the other $N - 1$ coefficients of g'.

3.3.3 Meet-in-the-Middle Attack

The search of the key space can be sped up using a combinatorial technique due to Odlyzko as presented by Howgrave-Graham, Silverman and Whyte in [18].

Binary Keys

We begin by describing the attack on a binary key f. The polynomial f has d_f coefficients equal to one and the rest of its coefficients are equal to zero. The polynomial g will have d_g ones and the rest of its entries will be zero. To simplify our exposition we will make the assumption that N and d_f are both even, but the attack is the same when either are odd.

We know there exists a rotation of f where exactly half the ones lie in the first $N/2$ entries. The idea is to search for f in the form $f = f_1||f_2$ where f_1 and f_2 are both polynomials of length $N/2$ with $d_f/2$ ones and $||$ denotes concatenation. We can think of f_1 as a polynomial of length N with the last $N/2$ coefficients zero and similarly we can think of f_2 as a polynomial of length N with the first $N/2$ coefficients zero, so that $f = f_1 + f_2$.

Notice that if

$$f * h \equiv g \mod q$$

then

$$\begin{aligned} (f_1||f_2) * h &\equiv g \mod q \\ f_1 * h &\equiv g - f_2 * h \mod q. \end{aligned}$$

Since g is a binary polynomial, this means that the coefficients of $f_1 * h$ and $f_2 * h$ all differ by either 0 or 1 modulo q.

If we find an $f' \in \mathcal{L}_f$ such that $g' \equiv f' * h \mod q$ is binary it is likely that we have found a rotation of f or another alternate key that is able to decrypt messages. The meet-in-the-middle attack terminates when we find an f' of this form.

The following is the algorithm for the meet-in-the-middle attack on binary keys presented in [18] for a positive integer $k < N$.

1. **Enumerate the vectors f_1.** There are $\binom{N/2}{d_f/2}$ polynomials of length $N/2$ with exactly $d_f/2$ ones and the rest of the entries 0. We put each such f_1 into a bin

corresponding to the most significant bit of the first k coordinates of the binary representation of $f_1 * h \mod q$. Each bin is then labeled by $\{0,1\}^k$ so there are a total of 2^k bins.

For example, suppose $N = 5$, $q = 8$ and $k = 3$. Next suppose $f_1 * h \mod q = 7 + 2x + 5x^2 + 3x^3 + 6x^4$. We are only interested in the first three coordinates. We can write each of their coordinates in binary notation

$$\begin{aligned} 7 &= (\underline{1}11)_2 \\ 2 &= (\underline{0}10)_2 \\ 5 &= (\underline{1}01)_2 \end{aligned}$$

so the polynomial f_1 would be put into the bin labeled $[1,0,1]$. Note when we reduce modulo q we are reducing to the interval $[0, q-1]$, however we could also reduce to $(-q/2, q/2]$ and take the sign as the most significant bit.

2. **Enumerate the vectors f_2.** There are also $\binom{N/2}{d_f/2}$ possible f_2's. For each such f_2 we check to see if $-f_2 * h$ corresponds to a bin occupied by an f_1. Meaning the leading bit of first k coefficients of $-f_2 * h \mod q$ are equal to $f_1 * h \mod q$. We want to find an f_1 and f_2 such that $(f_1 * h)_i \equiv \{0,1\} - (f_2 * h)_i \mod q$, therefore we not only want to check the bins that correspond to $-f_2 * h$ but we also want to check the bins that correspond to adding ones to any of the coefficients of $-f_2 * h$.

   For example, suppose $N = 5$, $q = 8$, $k = 3$ and the first three terms in $-f_2 * h$ are $6 + 3x + 7x^2$. The bin representing $-f_2 * h$ is $[1,0,1]$. If we add one to six we get seven, and the leading bit remains one. However if we add one to three we get four and the leading bit changes from zero to one. Similarly if we add one to seven we get zero so the leading bit changes from one to zero. So we need to check the bin $[1,0,1]$ as well as the bins $[1,1,1]$, $[1,1,0]$ and $[1,0,0]$.

3. **Check for Matches** When we find a bin corresponding to both an f_1 and an f_2 we know that the first k coefficients of $f * h \mod q$ are small where $f = f_1||f_2$, therefore it is possible $f * h \mod q$ is a binary polynomial as wanted. We compute $f = f_1||f_2$ and check to see if the coefficients of $f * h \mod q$ are binary. If it is binary we terminate the algorithm as we have likely found a private key. If it is not binary we test the next f_2. Note that if the bin is occupied by more than one f_1 we want to check f_2 with each f_1 in the bin.

Running Time and Memory

We now look at the running time and memory needed for the meet-in-the-middle attack. The calculations are my own and and are a slight improvement to those in [18].

The first step is to enumerate the f_1's and write each as to the bin corresponding to $f_1 * h \mod q$. Since there are $\binom{N/2}{d_f/2}$ possible f_1's, the expected time to enumerate all possible f_1's and write them to their corresponding bins is approximately

$$\tau_1 = \binom{N/2}{d_f/2}(\tau_c + \tau_l) \tag{3.3.3}$$

where τ_c is the time it takes to compute the convolution $f_1 * h \mod q$ and τ_l is the time it take to look up and write to a bin.

Next we need to enumerate the f_2's. Again there are a total of $\binom{N/2}{d_f/2}$ possible f_2's. For each f_2 we need to calculate $-f_2 * h \mod q$ and check the corresponding bins. If n out of the first k bits flip when we add one to the first k coefficients of $-f_2 * h$ we will have to check 2^n bins. It follows that the expected number of bins to check is $(2^c)^k$ where c is the probability the leading bit will flip when we add one. There are only two cases where the leading bit will change when we add one and there are q possible values for each coefficient hence $c = \frac{2}{q}$.

Now for each occupied bin corresponding to $-f_2 * h \mod q$ we must calculate $f * h \mod q$ where $f = f_1||f_2$. We make the assumption that each bin has the same

probability of being occupied. Since there are $\binom{N/2}{d_f/2}$ f_1's we will assume that $\binom{N/2}{d_f/2}/2^k$ is the proportion of bins that are occupied. Therefore the expected time for this step will be at most

$$\tau_2 = \binom{N/2}{d_f/2}(\tau_c + 2^{\frac{2k}{q}}\tau_l + 2^{\frac{2k}{q}}\frac{\binom{N/2}{d_f/2}}{2^k}\tau_c). \tag{3.3.4}$$

The total running time for the algorithm is approximately $\tau_1 + \tau_2$. If we choose k as in [18] we have $2^k \approx 100\binom{N/2}{d_f/2}$ so $|\binom{N/2}{d_f/2}/2^k| \approx 1/100$. Also note that $2^{\frac{2k}{q}} \approx 1$ for standard NTRU parameter choices. For example for NTRU parameter set $N = 401$, $d_f = 113$ and $q = 2048$ taken from the IEEE standard [21], we can take $k = 170$ and have $2^k \approx 100\binom{\lceil N/2 \rceil}{\lceil d_f/2 \rceil}$. In this case we have $2^{\frac{2k}{q}} \approx 1.12$. In practice the leading term in $\tau_1 + \tau_2$ is of order $\binom{N/2}{d_f/2}(\tau_c + \tau_l)$.

N	d_f	Combinatorial Security	Size of Key Space
43	14	1.16×10^5	2.29×10^{10}
83	27	5.29×10^{10}	5.10×10^{21}
107	35	9.69×10^{13}	1.94×10^{28}
401	113	7.05×10^{50}	1.60×10^{102}

Table 3.2: Comparison of combinatorial security and the size of the key space for NTRU private key f in the case where $p = 2$.

Therefore we need to search through approximately $\binom{N/2}{d_f/2}$ possibilities for the NTRU private key f. We say that

$$Comb_B = \binom{N/2}{d_f/2}$$

is the **combinatorial security** of the NTRU. In Table 3.2 we compare the combinatorial security with the size of the key space for the NTRU private key f given by expression 3.3.1. We see that the meet-in-the-middle attack cuts the search time on the key space by approximately a square root. Notice that all possible f's of the form $f = f_1||f_2$ are contained in $\mathcal{L}_f$, therefore

$$\binom{N/2}{d_f/2}\binom{N/2}{d_f/2} < \binom{N}{d_f}$$

and

$$\binom{N/2}{d_f/2} < \sqrt{\binom{N}{d_f}}$$

is an upper bound.

Next we consider the memory required to implement the meet-in-the-middle attack. Suppose μ_f is the memory required to label and store an f_1. Memory does not need to be allocated to store the f_1's into bins until it is needed if the bins are held in a linked list structure. This results in a reduction of memory needed, however it increases the time required to add and retrieve information from the bins. Hence the memory needed is not highly dependent on our choice of k; it is approximately $\binom{N/2}{d_f/2}\mu_f$.

Trinary Keys

Now we want to consider the attack on trinary keys. The running time for such an attack is given in [15], however they do not provide a derivation. Therefore we develop our own attack by modifying the attack on binary keys.

Recall in the case where $p = 3$ the NTRU keys f and g will have coefficients equal to 1, -1 and 0. To simplify our presentation we will assume f has d_f 1's and d_f -1's and we will assume N is even.

There exists a rotation of f that contains exactly half of its nonzero coefficients in the first $N/2$ terms. Therefore we will search for an f in the form $f = f_1||f_2$ where f_1 and f_2 are both polynomials of length $N/2$ with exactly d_f non-zero terms equal to 1 or -1. As in the binary case we can think of f_1 as a polynomial of length N with the last $N/2$ coefficients zero and similarly we can think of f_2 as a polynomial of length N with the first $N/2$ coefficients zero, so that $f = f_1 + f_2$.

We need to enumerate all possible vectors f_1. We can count the f_1's by first choosing the d_f nonzero positions. Out of these d_f positions we will then select the positions for the 1's. The -1's will automatically go in the remaining positions.

Hence the total possible number of f_1's is

$$\binom{N/2}{d_f}\left[\binom{d_f}{d_f}+\binom{d_f}{d_f-1}+\cdots+\binom{d_f}{0}\right].$$

Since $\sum_{i=0}^{n}\binom{n}{i}=2^n$ we have that the total number of f_1's (and f_2's) is

$$2^{d_f}\binom{N/2}{d_f}.$$

We will bin the f_1's in the same manner as we did for the binary case.

Next we need to enumerate the f_2's. We need to check each f_2 and see if it corresponds to an occupied bin. If $f_1 * h \equiv g - f_2 * h \mod q$, the coefficients of $f_1 * h$ and $-f_2 * h$ will differ by either 1, -1 or 0. Therefore we will also need to check the bins that correspond to adding 1, -1 or 0 to each of the first k coefficients of $-f_2 * h$.

When we find an occupied bin corresponding to f_2 we compute $f' = f_1||f_2$ and calculate $g' = f' * h \mod q$ and determine if g' is trinary. If g' is trinary we terminate the algorithm as we have likely found an alternate key that can decrypt messages. If not, we move on to the next f_2. As before, if there is more than one f_1 in an occupied bin we must check f_2 with each f_1.

Running Time and Memory for Trinary Keys

We now look at the running time an memory needed for the meet in the middle attack on trinary keys.

The first step is to enumerate the possible f_1's and write them to their corresponding bins. This will take approximately

$$\tau_1 = 2^{d_f}\binom{N/2}{d_f}(\tau_c+\tau_l)$$

steps where τ_c is the time it takes to compute a convolution product and τ_l is the time it take to look up and write to a bin.

Next we must enumerate the f_2's and check for occupied bins. In the trinary case there are two cases where the leading bit of $-f_2 * h$ changes when we add 1 and two

cases where the leading bit changes when we add -1. Hence the probability that the leading bit of $-f_2 * h$ changes when we add ± 1 is $\frac{4}{q}$. Therefore the expected number of bins we will need to check is $2^{\frac{4k}{q}}$. As before we will assume the f_1's are equally distributed among the bins hence the probability a bin is occupied is $\frac{2^{d_f}\binom{N/2}{d_f}}{2^k}$. Hence the expected number of steps will be at most

$$\tau_2 = 2^{d_f}\binom{N/2}{d_f}(\tau_c + 2^{\frac{4k}{q}}\tau_l + 2^{\frac{4k}{q}}\frac{2^{d_f}\binom{N/2}{d_f}}{2^k}\tau_c).$$

By similar arguments to the binary case, if we choose k such that $2^k \gg 2^{d_f}\binom{N/2}{d_f}$ the leading term in $\tau_1 + \tau_2$ is of the order $2^{d_f}\binom{N/2}{d_f}(\tau_c + \tau_c)$. Therefore the combinatorial security of NTRU for trinary keys is

$$Comb_T = 2^{d_f}\binom{N/2}{d_f}. \tag{3.3.5}$$

N	d_f	$Comb_T$	$Comb_{T2}$	Size of Key Space
43	14	5.24×10^9	2.31×10^{10}	6.08×10^{18}
83	27	1.34×10^{19}	1.12×10^{21}	3.76×10^{37}
107	35	6.31×10^{24}	1.12×10^{21}	8.35×10^{48}
401	113	3.98×10^{92}	2.02×10^{101}	4.59×10^{184}

Table 3.3: Comparison of combinatorial security and the size of the key space for NTRU private key f in the case where $p = 3$.

Note however, the combinatorial security for trinary keys given in [15] is

$$Comb_{T2} = \frac{\binom{N}{d_f+1}}{\sqrt{N}}.$$

This is likely due to a different method of choosing f_1 and f_2. For example one can let the set of f_1's be polynomials of length N with $d_f + 1$ 1's and the set of f_2's be the set of polynomials of length N with d_f -1's.

In Table 3.3 we compare $Comb_T$ with $Comb_{T2}$ and the size of the key space for the NTRU private key f given by expression 3.3.2. Our combinatorial security

$Comb_T$ appears to be lower than $Comb_{T2}$ therefore the attack we describe, leading to $Comb_T$ is the better choice for the attacker. We also see that the meet-in-the-middle attack on trinary keys we developed appears to also cut the search time on the key space by a square root.

Similarly since the expected number of occupied bins is $2^{d_f}\binom{N/2}{d_f}$ so the memory required is approximately

$$2^{d_f}\binom{N/2}{d_f}\mu_f$$

where μ_f is the memory required to store and label f.

3.3.4 Lattice Attacks

Coppersmith and Shamir proposed a lattice attack on the NTRU private key based on the SVP in [4].

The standard NTRU lattice $\mathcal{L}^{NT}$ is the lattice of dimension $2N$ generated by the rows of the following $2N \times 2N$ matrix:

$$\mathcal{B}^{NT} = \left[\begin{array}{cccc|cccc} \lambda & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ 0 & \lambda & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda & h_1 & h_2 & \cdots & h_0 \\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right]$$

where $h = \sum h_i x^i$ is the NTRU public key and $\lambda \in \mathbb{R}$ is some nonzero constant. (How the attacker selects the value of λ will be discussed at the end of this section.) A more convenient way to write this matrix is:

$$\mathcal{B}^{NT} = \begin{bmatrix} \lambda I & H \\ 0 & qI \end{bmatrix}$$

where I is the $N \times N$ identity matrix and H is the **circulant matrix** of h. This means that the matrix multiplication $xH = x * h$ for any polynomial $x \in \mathcal{R}$. We show that the vector $(\lambda f \quad g)$ is contained in the lattice $\mathcal{L}^{NT}$.

Recall that $h = F_q * g \mod q$ or equivalently that $f * h \equiv g \mod q$. We can rewrite this expression as $f * h = g + q * u$ for some polynomial $u \in \mathcal{R}$. Notice that:

$$v' = \begin{bmatrix} f & -u \end{bmatrix} \begin{bmatrix} \lambda I & H \\ 0 & qI \end{bmatrix} = \begin{bmatrix} \lambda f & f * h - q * u \end{bmatrix} = \begin{bmatrix} \lambda f & g \end{bmatrix} \in \mathcal{L}^{NT}.$$

Therefore the vector $(\lambda f \quad g)$ is contained in the lattice. If we can recover the vector $(\lambda f \quad g)$ then we can recover the NTRU private key f. Note, by similar arguments, all the vectors in the lattice $\mathcal{L}^{NT}$ are of the form $(\lambda f' \quad g')$ where $g' \equiv f' * h \mod q$.

Since the coefficients of the polynomials f and g all have norm 0 or 1 the vector $(\lambda f \quad g)$ will be among the short vectors in the lattice $\mathcal{L}^{NT}$. Hence we can use lattice reduction techniques such as the LLL or BKZ algorithms to try and recover the vector $(\lambda f \quad g)$ or other short vectors of the form $(\lambda f' \quad g')$ such as rotations of f and g.

Proposition 3.3.1 *Suppose $p = 3$ and suppose f and g are trinary keys. The vector $(\underbrace{\lambda, \ldots, \lambda}_{N}, \underbrace{0, \ldots, 0}_{N})$ is contained in the NTRU lattice $\mathcal{L}^{NT}$.*

Proof: Any vector contained in the lattice generated by $\mathcal{B}^{NT}$ can be expressed as an integer linear combination of the rows in $\mathcal{B}^{NT}$. Notice that if we add the first N rows of $\mathcal{B}^{NT}$ we get the vector

$$(\underbrace{\lambda, \ldots, \lambda}_{N}, \underbrace{k, \ldots, k}_{N})$$

where k is the sum of the coefficients in the polynomial h. This means we only need to show that the sum of the coefficients of h is $0 \mod q$.

We have that $f * h \equiv g \mod q$ or $h \equiv F_q * g \mod q$. Hence $h = F_q g + u(x^N - 1)$ for some $u \in \mathbb{Z}/q\mathbb{Z}[x]$. In the trinary case the g has d_g 1's and d_g -1's so $g(1) = 0$. Therefore $h(1) \equiv F_q(1)g(1) + u(1)(0) \equiv 0 \mod q$. This means the coefficients of h

add to zero modulo q. ∎

Therefore the vector $(\underbrace{\lambda, \ldots, \lambda}_{N}, \underbrace{0, \ldots, 0}_{N})$ is another short vector in the lattice. This vector may be shorter that our target vector $(\lambda f \quad g)$, as in the case where we take $d_f = d_g = \lfloor N/3 \rfloor$. However, experiments seem to show this does not greatly effect the efficiency of LLL of finding our target vector.

Balancing Constant λ

The constant λ is a balancing constant which is chosen to maximize the efficiency of the search for small vectors in the lattice [9], [14].

Recall from Section 2.3.2 that lattice reduction techniques will have the best chance of finding the target vector of length τ, or a vector whose length is close to τ if τ is much shorter than the shortest expected vector in the lattice. Hence we want to choose λ to maximize the ratio s/τ where s is the length of the shortest expected vector in the lattice. Also recall from expression 2.3.3 that the shortest non-zero vector in a lattice $\mathcal{L}$ given by the Gaussian Heuristic is

$$s = \sqrt{\frac{d}{2\pi e}} (\det \mathcal{L})^{1/d}$$

where d is the dimension of the lattice. The dimension of $\mathcal{L}^{NT}$ is $d = 2N$ and $\det \mathcal{L}^{NT} = \lambda^N q^N$, hence the expected length of the shortest non-zero vector in $\mathcal{L}^{NT}$ is

$$s = \sqrt{\frac{N\lambda q}{\pi e}}.$$

The length of our target vector $(\lambda f \quad g)$ is

$$\tau = \sqrt{\lambda^2 \|f\|^2 + \|g\|^2}.$$

Therefore we want to maximize the ratio

$$s/\tau = \sqrt{\frac{Nq}{\pi e}} \sqrt{\frac{\lambda}{\lambda^2 \|f\|^2 + \|g\|^2}}.$$

This is the same as maximizing $\lambda/(\lambda^2 \|f\|^2 + \|g\|^2) = (\lambda \|f\|^2 + \lambda^{-1} \|g\|^2)^{-1}$. One can see the derivative with respect to λ of $(\lambda \|f\|^2 + \lambda^{-1} \|g\|^2)^{-1}$ is

$$-(\lambda \|f\|^2 + \lambda^{-1} \|g\|^2))^{-2}(\|f\|^2 - \lambda^{-2} \|g\|^2),$$

hence to maximize the ratio we need $\|f\|^2 - \lambda^{-2} \|g\|^2 = 0$ or $\lambda = \|g\| / \|f\|$. One can easily check this gives a maximum.

In practice if $d_f = d_g$ then $\|f\| \approx \|g\|$ and we take $\lambda = 1$.

3.4 NTRU Over Euclidean Domains

We have seen that the NTRU cryptosystem is constructed over polynomial rings with coefficients in $\mathbb{Z}$. To further improve the security and efficiency of NTRU we now look at extending NTRU to other rings as has been explored in [3], [24], [25] and [29].

Let A be a ring and set $\mathcal{R} = A[x]/\langle x^N - 1\rangle$. Let $p, q \in A$. To follow the NTRU encryption and decryption algorithms we need to be able to reduce the coefficients of a polynomial in $\mathcal{R}$ modulo p and q. For any $a, r \in A$ we have $a \equiv r \mod p$ if and only if $a = bp + r$ for some $b \in A$. Therefore to perform reduction modulo p and q it would suffice to have a division algorithm over the ring A.

Recall that an **integral domain** is a commutative ring with identity $1 \neq 0$ that has no nonzero zero-divisors. Examples include fields and $\mathbb{Z}$.

Definition 3.4.1 *An integral domain A is an **Euclidean domain** if there exists a function $d : A \longrightarrow \mathbb{Z}_{\geq 0}$ that satisfies*

(1) for any nonzero $a, b \in A$, $d(a) \leq d(ab)$, and

(2) for any $a, b \in A$ with $b \neq 0$ there exist elements $q, r \in A$ such that

$$a = qb + r$$

where either $r = 0$ or $d(r) < d(b)$.

Definition 3.4.2 *Suppose A is an Euclidean domain. We define a* ***division algorithm*** *to be any process which outputs a value $r \in A$ when applied to a pair $(a, b) \in A \times A \setminus \{0\}$ as in Definition 3.4.1 (2).*

We say that the division algorithm outputs a unique representative of the congruence classes if for all $b \in A \setminus \{0\}$ and for all $a, a' \in A$ such that $a \equiv a' \mod b$, the output of the division algorithm on (a, b) and (a', b) is the same.

Notice that if a division algorithm outputs r on the input (a, b) then $r \equiv a \mod b$. Given a division algorithm which assigns (a, b) to r we write

$$r = a \mod b.$$

We say that $a \in A$ is **reduced modulo** b if the output of the division algorithm applied to the pair (a, b) is a.

Unfortunately not every Euclidean domain has a known division algorithm.

Setup for an "NTRU-like" cryptosystem:

Let A be an Euclidean domain with a division algorithm which outputs unique representatives of the congruence classes. We define the NTRU parameters as follows: Let N be a positive integer, let $p, q \in A$ and let $\mathcal{R} = A[x]/\langle x^N - 1\rangle$. Take $f, g, \phi, m \in \mathcal{R}$ with coefficients reduced modulo p, such that there exists elements $F_p, F_q \in \mathcal{R}$ satisfying the following properties:

$$F_p * f \equiv 1 \mod p \quad \text{and} \quad F_q * f \equiv 1 \mod q.$$

Set $h = F_q * g \mod q$.

To encrypt a message m following the NTRU encryption algorithm we set

$$e = p\phi * h + m \mod q.$$

To decrypt a ciphertext e following the NTRU decryption algorithm first we set

$$a = f * e \mod q,$$

next we set

$$m' = F_p * e \mod p.$$

Theorem 3.4.3 *Let A be an Euclidean domain. Suppose that there exists a division algorithm that outputs unique representatives of the congruences classes. Then NTRU decryption algorithm recovers the original message m (that is $m' = m$), if all of the coefficients of polynomial $p\phi * g + f * m$ are reduced modulo q.*

Proof: To encrypt a message m following the NTRU encryption algorithm we set $e = p\phi * h + m \mod q$. To decrypt e first we set

$$\begin{aligned} a &= f * e \mod q \\ &\equiv pf * \phi * h + f * m \mod q. \end{aligned}$$

Under the assumption that $a = p\phi * h + f * m$ we have

$$\begin{aligned} F_p * a &\equiv F_p * (p\phi * g + f * m) \mod p \\ &\equiv pF_p * \phi * g + m \mod p \\ &\equiv m \mod p. \end{aligned}$$

Since m has coefficients reduced modulo p and the division algorithm outputs unique representatives of the congruence classes modulo p we have recovered our message m when we reduce the coefficients of $F_p * a$ modulo p. ∎

To increase the probability that all of the coefficients of the polynomial $p\phi * h + f * m$ are reduced modulo q we want a division algorithm that outputs the 'smallest' representatives of the congruences classes modulo q.

3.4.1 Gaussian Integers as an NTRU Base Ring

The ring of **Gaussian integers**, denoted $\mathbb{Z}[i]$, is the set of complex numbers of the form $a+bi$, where a and b are integers. In his thesis [25] Kouzmenko suggests using the

Gaussian integers for the NTRU base ring A. In this section we present Kouzmenko's division algorithm for the Gaussian integers and show that the Gaussian integers are a Euclidean domain. In the next section we extend Kouzmenko's work to lattices.

For $\alpha = a + bi \in \mathbb{Z}[i]$ we define $d(\alpha) = a^2 + b^2$ which is the square of the complex norm $|\alpha| = \sqrt{a^2 + b^2}$. Since for any two complex numbers α and β we have $|\alpha\beta| = |\alpha|\,|\beta|$ if follows that $d(\alpha\beta) = d(\alpha)d(\beta)$. Therefore the first condition for a Euclidean domain is satisfied as for any nonzero $\alpha, \beta \in \mathbb{Z}[i]$ we have $d(\alpha) \leq d(\alpha)d(\beta) = d(\alpha\beta)$ since $d(\beta) > 1$. Next we show there exists a division algorithm, which implies $\mathbb{Z}[i]$ is a Euclidean domain.

The following theorem presented by Kouzmenko shows that there exists a division algorithm.

Theorem 3.4.4 *[25, Theorem 17] If α,$\beta \in \mathbb{Z}[i]$ and $\beta \neq 0$, then there exists an algorithm to find γ,$\rho \in \mathbb{Z}[i]$ such that*

$$\alpha = \gamma\beta + \rho$$

with $d(\rho) \leq \frac{1}{2}d(\beta)$.

Proof: This a constructive proof given by Kouzmenko in [25] which gives us an algorithm to find such a γ and ρ.

Suppose that α,$\beta \in \mathbb{Z}[i]$ and $\beta \neq 0$. Let $\alpha = a + bi$ with $a, b \in \mathbb{Z}$. Let us first consider the case where $\beta = n$ is a positive integer. Using the Euclidean division algorithm applied the ring of integers we are able to find integers $a_\gamma, a_\rho, b_\gamma$ and b_ρ such that:

$$\begin{aligned} a &= a_\gamma n + a_\rho \\ b &= b_\gamma n + b_\rho \end{aligned}$$

with $-\frac{1}{2}n < a_\rho \leq \frac{1}{2}n$ and $-\frac{1}{2}n < b_\rho \leq \frac{1}{2}n$. We define $\gamma, \rho \in \mathbb{Z}[i]$ as:

$$\gamma = a_\gamma + b_\gamma i \quad \text{and}$$

$$\rho \;=\; a_\rho + b_\rho i.$$

Hence $\alpha = a + bi = a_\gamma n + a_\rho + (b_\gamma n + b_\rho)\, i = (a_\gamma + b_\gamma i)\, n + (a_\rho + b_\rho i) = \gamma n + \rho$. We note that $d(\rho) = a_\rho^2 + b_\rho^2 \leq \frac{n^2}{4} + \frac{n^2}{4} = \frac{1}{2} d(n)$. Thus the theorem holds in this special case.

Now let β be any non-zero Gaussian integer. Then $n = \bar{\beta}\beta$ is a positive integer. We are then able to apply the above algorithm to $\bar{\beta}\alpha$. That is we can find Gaussian integers γ and ρ' such that $\bar{\beta}\alpha = \gamma n + \rho'$ with $\rho' = 0$ or $d(\rho') \leq \frac{1}{2} d(n)$. Since $\rho' = \bar{\beta}\alpha - \gamma n$ and $n = \bar{\beta}\beta$ we have

$$d(\rho') = d\left(\bar{\beta}\alpha - \gamma\bar{\beta}\beta\right) \leq \frac{1}{2} d\left(\bar{\beta}\beta\right)$$

and since $d(xy) = d(x)\, d(y)$ for any Gaussian integers x and y we get

$$d\left(\bar{\beta}\right) d(\alpha - \gamma\beta) \leq \frac{1}{2} d\left(\bar{\beta}\right) d(\beta)$$

which means

$$d(\alpha - \gamma\beta) \leq \frac{1}{2} d(\beta).$$

Let $\rho = \alpha - \gamma\beta$ then $\alpha = \gamma\beta + \rho$ with γ and ρ Gaussian integers and $d(\rho) \leq \frac{1}{2} d(\beta)$ as wanted. ∎

The proof of this theorem gives us a division algorithm to calculate a representative of α modulo β. That is if the algorithm outputs $\alpha = \gamma\beta + \rho$ then $\alpha \equiv \rho \mod \beta$.

To be able to use Gaussian integers for NTRU we need to make sure that the algorithm chooses a unique representative of each equivalence class modulo β and that the representative chosen is the representative with the smallest norm. For example suppose $\beta = 2 + 3i$. Then

$$\alpha_1 \;=\; -1 - 2i = -\beta + (1 + i)$$

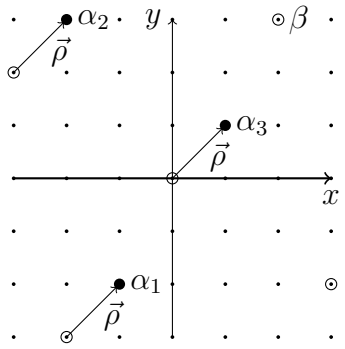

Figure 3.2: The open circles represent multiples $\gamma\beta$ of $\beta = 2 + 3i$ with $\gamma \in \mathbb{Z}[i]$. The Gaussian integers α_1, α_2 and α_3, indicated with solid dots, are all equivalent to ρ modulo β.

$$\begin{aligned} \alpha_2 &= -2 + 4i = i\beta + (1 + i) \quad \text{and} \\ \alpha_3 &= 1 + i = 0\beta + (1 + i) \end{aligned}$$

are all equivalent modulo β (see Figure 3.2). We want the division algorithm to always output the same representative for an equivalence class. By our analogies of NTRU we also want the division algorithm to always output the smallest representative $\rho = \alpha_3 = 1 + i$. The following theorem will prove the uniqueness of the division algorithm.

Theorem 3.4.5 *[25, Theorem 18] The division algorithm described in the proof of the Theorem 3.4.4 outputs unique representatives modulo β. That is suppose α_1,α_2 and β are three Gaussian integers with $\beta \neq 0$ and $\alpha_1 \equiv \alpha_2 \mod \beta$. If the algorithm outputs ρ_1 and ρ_2 with $d(\rho_1) \leq \frac{d(\beta)}{2}$ and $d(\rho_2) \leq \frac{d(\beta)}{2}$ such that*

$$\begin{aligned} \alpha_1 &= \gamma_1\beta + \rho_1 \\ \alpha_2 &= \gamma_2\beta + \rho_2, \end{aligned}$$

then $\rho_1 = \rho_2$.

Proof: Let α_1,α_2 and β be Gaussian integers. Suppose that $\beta \neq 0$ and that

$\alpha_1 \equiv \alpha_2 \mod \beta$. Then the division algorithm gives us:

$$\begin{aligned} \alpha_1 &= \gamma_1\beta + \rho_1, \\ \alpha_2 &= \gamma_2\beta + \rho_2. \end{aligned}$$

We need to show that $\rho_1 = \rho_2$

We will begin with the case that $\beta = n$ is an integer. Let $\alpha_1 = a_1 + b_1 i$. Then the division algorithm applied to the ring of integers gives us:

$$\begin{aligned} a_1 &= a_{1\gamma}n + a_{1\rho} \\ b_1 &= b_{1\gamma}n + b_{1\rho} \end{aligned}$$

with $-\frac{1}{2}n < a_{1\rho} \leq \frac{1}{2}n$ and $-\frac{1}{2}n < b_{1\rho} \leq \frac{1}{2}n$. Next we set:

$$\begin{aligned} \gamma_1 &= a_{1\gamma} + b_{1\gamma}i \\ \rho_1 &= a_{1\rho} + b_{1\rho}i \end{aligned}$$

so $\alpha_1 = \gamma_1\beta + \rho_1$ with $d(\rho_1) \leq \frac{d(\beta)}{2}$. Since $\alpha_1 \equiv \alpha_2 \mod \beta$ we can write α_2 as:

$$\alpha_2 = \delta n + \rho_1 = (cn + a_{1\rho}) + (dn + b_{1\rho})\, i = a_2 + b_2 i$$

for some Gaussian integer $\delta = c + di$. We can now apply the division algorithm to α_2 and we get

$$\begin{aligned} a_2 &= a_{2\gamma}n + a_{2\rho} \\ b_2 &= b_{2\gamma}n + b_{2\rho} \end{aligned}$$

since the division algorithm over the integers outputs a unique $a_{2\rho}$ and $b_{2\rho}$ with $-\frac{1}{2}n < a_{2\rho} \leq \frac{1}{2}n$ and $-\frac{1}{2}n < b_{2\rho} \leq \frac{1}{2}n$ and we have that $-\frac{1}{2}n < a_{1\rho} \leq \frac{1}{2}n$ and $-\frac{1}{2}n < b_{1\rho} \leq \frac{1}{2}n$ we get $a_{2\gamma} = c$, $a_{2\rho} = a_{1\rho}$, $b_{2\gamma} = d$ and $b_{2\rho} = b_{1\rho}$. Hence

$$\begin{aligned} \gamma_2 &= a_{2\gamma} + b_{2\gamma}i = c + di \\ \rho_2 &= a_{2\rho} + b_{2\rho}i = a_{1\rho} + b_{1\rho}i. \end{aligned}$$

Hence $\rho_1 = \rho_2$ which proves the theorem in this case.

Now suppose β is any non-zero Gaussian integer. We set $n = \beta\bar{\beta}$. Since $\alpha_1 \equiv \alpha_2 \mod \beta$ we have that $\alpha_1\bar{\beta} \equiv \alpha_2\bar{\beta} \mod \beta$. We apply the division algorithm above to get:

$$\begin{aligned} \alpha_1\bar{\beta} &= \gamma_1 n + \rho_1' \\ \alpha_2\bar{\beta} &= \gamma_2 n + \rho_2' \end{aligned}$$

From the first part of the proof we know that $\rho_1' = \rho_2'$. The algorithm outputs $\rho_1 = \gamma_1 - \rho_1'\beta$ and $\rho_2 = \gamma_2 - \rho_2'\beta$. We need to show that $\rho_1 = \rho_2$. Since $\rho_1' = \rho_2'$ and $\beta \neq 0$ we have

$$\alpha_1\bar{\beta} - \gamma_1 n = \alpha_2\bar{\beta} - \gamma_2 n$$

and since $\beta\bar{\beta} = n$ we have

$$\begin{aligned} \alpha_1\bar{\beta} - \gamma_1\beta\bar{\beta} &= \alpha_2\bar{\beta} - \gamma_2\beta\bar{\beta} \\ \alpha_1 - \gamma_1\beta &= \alpha_2 - \gamma_2\beta \end{aligned}$$

Hence $\rho_1 = \rho_2$. ∎

3.4.2 The Gaussian Integers as a Lattice

Note that the Gaussian integers $\mathbb{Z}[i]$ form a square lattice generated by the basis $B = \{1, i\}$. Let β be a Gaussian integer. We define

$$\begin{aligned} \mathcal{L}(\beta) &= \{\gamma\beta | \gamma \in \mathbb{Z}[i]\} \\ &= \{(a + bi)\beta | a, b \in \mathbb{Z}\} \\ &= \{a\beta + b\beta i | a, b \in \mathbb{Z}\} \end{aligned}$$

hence $\mathcal{L}(\beta)$ is also a square lattice generated by the basis $\{\beta, i\beta\}$. Recall that the complex norm $\|\alpha\| = \sqrt{d(\alpha)}$.

In fact, the division algorithm presented by Kouzmenko is a closest vector algorithm for the lattice $\mathcal{L}(\beta)$. We claim the algorithm outputs $\rho = \alpha - \gamma\beta$ as a smallest representative of α modulo β. Hence the vector $\gamma\beta \in \mathcal{L}(\beta)$ minimizes the norm $\|\alpha - \omega\|$ that is $\gamma\beta$ is a closest vector to α on the lattice generated by β. The following theorem will show that the division algorithm does indeed output ρ as a smallest representative of α modulo β.

Theorem 3.4.6 *The division algorithm described in the proof of Theorem 3.4.4 is a closest vector algorithm. That is, it finds a $\gamma \in \mathbb{Z}[i]$ such that $\gamma\beta$ is a closest vector to α in $\mathcal{L}(\beta)$.*

Proof: We want to find the closest vector to α in the lattice $\mathcal{L}(\beta)$. Recall that the lattice $\mathcal{L}(\beta)$ is generated by the basis $\{\beta, \beta i\}$. This gives a rectangular lattice.

In Section 2.2 we saw that it is easy to solve the CVP on a rectangular lattice. We transform the basis by multiplying by $\bar{\beta}$. We can easily find the closest vector to $\bar{\beta}\alpha = a + bi$, with $a, b \in \mathbb{Z}$, on the lattice generated by $\left\{\bar{\beta}\beta, \bar{\beta}\beta i\right\} = \{n, ni\}$ where $\bar{\beta}\beta = n \in \mathbb{Z}$ by rounding off the horizontal and vertical coordinates. That is let $\rho' = a_{\rho'} + b_{\rho'} i$ where

$$a_{\rho'} = a - n\lfloor a/n \rceil \quad \text{and} \quad b_{\rho'} = b - n\lfloor b/n \rceil$$

and let $\gamma' = a_{\gamma'} + b_{\gamma'} i$ where

$$a_{\gamma'} = n\lfloor a/n \rceil \quad \text{and} \quad b_{\gamma'} = n\lfloor b/n \rceil .$$

Hence γ' is the closest vector to $\bar{\beta}\alpha$ in the lattice generated by $\{n, ni\}$ and $\rho' = \bar{\beta}\alpha - \gamma'$. If we let $\gamma = \lfloor a/n \rceil + \lfloor b/n \rceil i$ then $\gamma' = n\gamma = \bar{\beta}\beta\gamma$ and so $\rho' = \bar{\beta}\alpha - \bar{\beta}\beta\gamma$. To return to our original lattice we multiply by $\bar{\beta}^{-1}$. Therefore $\rho = \bar{\beta}^{-1}\rho' = \alpha - \gamma\beta$.

Equivalently we have

$$a = a_\gamma n + a_{\rho'} \quad \text{and}$$

$$b = b_\gamma n + b_{\rho'}$$

with $-\frac{1}{2} < a_{\rho'} \leq \frac{1}{2}$, $-\frac{1}{2} < b_{\rho'} \leq \frac{1}{2}$ then $a_\gamma = \lfloor a/n \rceil$ and $b_\gamma = \lfloor b/n \rceil$. Set $\gamma = a_\gamma + b\gamma i$ and $\rho = \alpha - \gamma\beta$. This is exactly the algorithm given in Theorem 3.4.4. Hence the division algorithm presented by Kouzmenko is a closest vector algorithm and ρ is a smallest representative of the congruence class modulo β. ∎

We have shown that the division algorithm outputs the smallest and unique representatives for the congruence classes. We define a **fundamental domain** of $\mathcal{L}(\beta)$ to be a subset of $\mathbb{Z}[i]$ that contains a unique representative from all the congruence classes modulo β. Note that the intersection of $\mathbb{Z}[i]$ with the fundamental parallelepiped of $\mathcal{L}(\beta)$ described in Section 2.1 is one of many fundamental domains of $\mathcal{L}(\beta)$.

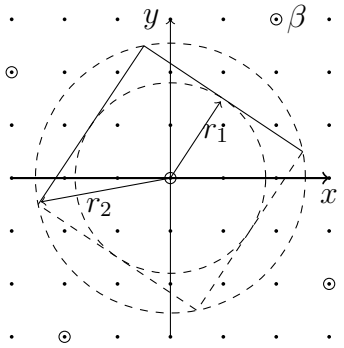

Figure 3.3: The open dots represent the lattice $\mathcal{L}(\beta)$ with $\beta = 2 + 3i$. The square represents a fundamental domain of $\mathcal{L}(\beta)$. The inscribed circle has radius $r_1 = \frac{\sqrt{d(\beta)}}{2}$ and the outer circle has radius $r_2 = \sqrt{\frac{d(\beta)}{2}}$

For example the division algorithm for reduction modulo β maps the Gaussian integers to a fundamental domain of $\mathcal{L}(\beta)$. The points inside the square in Figure 3.3 shows the fundamental domain given by the division algorithm. The edges of the square are equal distance from exactly two points in the lattice $\mathcal{L}(\beta)$ and the vertices

of the square are equidistant from exactly four points in the lattice $\mathcal{L}(\beta)$. Therefore, it is clear that the Gaussian integers inside the square are closest to the lattice point 0 and are the smallest representatives of the congruences classes modulo β. The opposite edges of the square are equivalent modulo β therefore we only include one of each opposite edge in the fundamental domain. One can apply the division algorithm to see which edge to include; it follows from the choice of the round off of 0.5 in the nearest integer function.

Recall that Kouzmenko proved that the division algorithm outputs a representative ρ modulo β with $d(\rho) \leq d(\beta)/2$ which is equivalent to $\|\rho\| \leq \sqrt{d(\beta)/2}$. That is he proved that ρ lies within the outer circle in Figure 3.3.

3.4.3 Extending NTRU over the Gaussian Integers

Kouzmenko has found a division algorithm for the Gaussian integers that outputs unique (and smallest) representatives of the congruence classes modulo β for $\beta \in \mathbb{Z}[i]$. This gave him all the elements needed to extend NTRU over the Gaussian integers (GNTRU).

Set $\mathcal{R} = \mathbb{Z}[i][x]/\langle x^N - 1\rangle$ and define the NTRU parameters in Section 3.1.1. The polynomials f, g and ϕ are chosen as before, however they have d_f, d_g and d_ϕ of each element from the set $\{1, -1, i, -i\}$. Since we need f to be invertible we take f to have $d_f + 1$ 1's.

In his thesis Kouzmenko developed a model for the probability of decryption failure for GNTRU. He compared NTRU and GNTRU for parameter sets with comparable key space sizes and probabilities of decryption failures. He concluded that the performance of GNTRU is comparable to that of NTRU in terms of speed and storage needed.

Next we extend NTRU over the Eisenstein integers which has a denser lattice than the Gaussian integers. We show that ETRU is an improvement to NTRU in

terms of both lattice security and storage and comparable to NTRU in terms of speed.

Chapter 4

NTRU over the Eisenstein Integers

In [24], KarimianPour suggests using Eisenstein integers for the NTRU base ring A (ETRU). The idea is extended by Nevins, KarimianPour and Miri in [29] where they give a division algorithm for the Eisenstein integers and propose a lattice for ETRU. In Section 4.1 we present a modified version of the division algorithm given in [29]. In Section 4.2 we state important properties of the Eisenstein integers and classify all the Eisenstein primes. In Section 4.3 we determine the number of distinct congruence classes modulo an Eisenstein integer. Finally in Section 4.4 we discuss parameter selection and the probability of decryption failure for ETRU.

4.1 A Division Algorithm over the Eisenstein Integers

We denote by ω a complex cube root of unity, that is $\omega^3 = 1$ where $\omega = \frac{1}{2}\left(-1+\sqrt{3}i\right)$. Since $\omega^3 - 1 = (\omega-1)(\omega^2+\omega+1) = 0$, we have $\omega^2+\omega+1 = 0$ and hence $\omega^2 = -1-\omega$. The ring of **Eisenstein integers**, denoted $\mathbb{Z}[\omega]$, is the set of complex numbers of the form $\alpha = a + b\omega$ with $a, b \in \mathbb{Z}$.

For $\alpha = a+b\omega$ we will define $d(\alpha) = \alpha\bar{\alpha} = a^2+b^2-ab$ which is the square of the

usual analytic complex norm $|\alpha|$. Note that $d(\alpha) = a^2 + b^2 - ab$ is a positive integer for $\alpha \neq 0$ since $d(\alpha)$ is the square of a norm and $a, b \in \mathbb{Z}$. For any complex numbers α and β we have that $|\alpha\beta| = |\alpha|\,|\beta|$ hence it follows that $d(\alpha\beta) = d(\alpha)\,d(\beta)$. Note that $d(\alpha)$ is sometimes called the algebraic norm of α.

The Eisenstein integers $\mathbb{Z}[\omega]$ form a lattice in $\mathbb{C}$ generated by the basis $B = \{1, \omega\}$. Note that the angle between the two basis vectors 1 and ω, represented by the vectors $(1, 0)$ and $(-1/2, \sqrt{3}/2)$ in $\mathbb{R}^2$, is 120 degrees and the basis vectors have equal length. Hence the Eisenstein integers form a hexagonal lattice as illustrated in Figure 4.1. Let β be an Eisenstein integer. We define the ideal $\mathcal{L}(\beta)$ by:

$$\begin{aligned}\mathcal{L}(\beta) &= \{\gamma\beta | \gamma \in \mathbb{Z}[\omega]\} \\ &= \{(a + b\omega)\beta | a, b \in \mathbb{Z}\} \\ &= \{a\beta + b\beta\omega | a, b \in \mathbb{Z}\}.\end{aligned}$$

Therefore $\mathcal{L}(\beta)$ is a lattice generated by the basis $\{\beta, \beta\omega\}$. Since multiplication by any $\beta \in \mathbb{C}$ preserves the angle between vectors, it is easy to verify that $\mathcal{L}(\beta)$ is also a hexagonal lattice.

We will now develop an Euclidean division algorithm for elements of $\mathbb{Z}[\omega]$ based on its lattice structure [29]. For any two elements $\alpha, \beta \in \mathbb{Z}[\omega]$ we need to find a $\gamma, \rho \in \mathbb{Z}[\omega]$ such that

$$\alpha = \gamma\beta + \rho$$

with $d(\rho) < d(\beta)$.

Recall the division algorithm for the Gaussian integers given in the proof of Theorem 3.4.4 is a closest vector algorithm. Consequently we want to develop a closest vector algorithm for the Eisenstein integers. First note that we are able to partition the Eisenstein integers as seen in Figure 4.1. The points represent the Eisenstein integers. The solid circles represent a rectangular sublattice $\mathcal{L}_R$ of $\mathbb{Z}[\omega]$ generated by $\{1, 1 + 2\omega\}$ and the open circles represent its coset $\mathcal{L}_R + \omega$. Recall in

Section 2.2 we saw that it is easy to solve the closest vector problem on a rectangular lattice. Therefore we are able to find a closest Eisenstein integer to an arbitrary complex number δ by finding a closest vector on $\mathcal{L}_R$ and a closest vector on $\mathcal{L}_R + \omega$ and take the closer of the two. Note that a closest vector to α on $\mathcal{L}_R + \omega$ can be found by finding a closest vector to $\alpha - \omega$ on $\mathcal{L}_R$ and adding ω to the result.

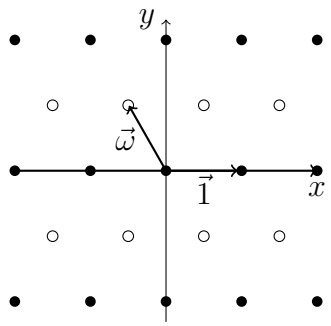

Figure 4.1: The circles represent the Eisenstein integers which are generated by the basis $\{1, \omega\}$. The Eisenstein integers can be partitioned into two subsets denoted by the solid and open circles.

The following theorem is a modified version of [29, Theorem 5.2] and gives a closest vector algorithm for $\mathcal{L}(\beta)$. The algorithm presented in [29] does not always output a unique representative for each congruence class.

Theorem 4.1.1 *[Division Algorithm] Let $\beta \in \mathbb{Z}[\omega] \setminus \{0\}$ and $\alpha \in \mathbb{Z}[\omega]$. Define $a_1, a_2, b_1, b_2 \in \mathbb{Q}$ by*

$$a_1 + b_1 i = \beta^{-1}\alpha \quad \textit{and} \quad a_2 + b_2 i = \beta^{-1}\alpha - \omega.$$

For $j = 1, 2$, compute

$$\rho'_j = (a_j - \lfloor a_j \rceil) + i\left(b_j - \sqrt{3}\left\lfloor b_j/\sqrt{3} \right\rceil\right).$$

Define $\rho_1, \rho_2, \gamma_1, \gamma_2 \in \mathbb{Z}[\omega]$ by:

$$\rho_1 \;=\; \beta\rho'_1 \quad \gamma_1 = \lfloor a_1 \rceil + i\sqrt{3}\left\lfloor b_1/\sqrt{3} \right\rceil \qquad \textit{and}$$

$$\rho_2 \;=\; \beta\rho_2' \quad \gamma_2 = \lfloor a_2 \rceil + i\sqrt{3}\left\lfloor b_2/\sqrt{3}\right\rceil + \omega.$$

Then the following hold:

$$\alpha = \beta\gamma_1 + \rho_1 = \beta\gamma_2 + \rho_2, \quad d(\rho_1), d(\rho_2) < d(\beta) \quad \textit{and} \quad Re(\gamma_1) \neq Re(\gamma_2),$$

where $Re(\gamma_1)$ denotes the real part of γ. Define (ρ, γ) by the following:

- *If $d(\rho_1) < d(\rho_2)$, set $(\rho, \gamma) = (\rho_1, \gamma_1)$;*
- *if $d(\rho_1) > d(\rho_2)$, set $(\rho, \gamma) = (\rho_2, \gamma_2)$;*
- *if $d(\rho_1) = d(\rho_2)$ and $Re(\gamma_1) > Re(\gamma_2)$ set $(\rho, \gamma) = (\rho_1, \gamma_1)$;*
- *if $d(\rho_1) = d(\rho_2)$ and $Re(\gamma_1) < Re(\gamma_2)$ set $(\rho, \gamma) = (\rho_2, \gamma_2)$,*

and process which outputs ρ when applied to the pair (α, β) is a division algorithm. Then the following hold:

1. *$\alpha = \beta\gamma + \rho$ and $d(\rho) < d(\beta)$*
2. *$\gamma\beta$ is an element of $\mathcal{L}(\beta)$ closest to α*
3. *ρ is a smallest representative of the congruence class modulo β*
4. *the division algorithm outputs a unique representative of the congruence class, that is if $\alpha, \alpha' \in \mathbb{Z}[\omega]$ satisfy $\alpha' \equiv \alpha \mod \beta$ then the algorithm produces the same output when applied to (α, β) and (α', β).*

Proof: First we check that $\alpha = \beta\gamma + \rho$ in both cases by expanding the right hand side. In the case where we set $\rho = \rho_1 = \beta\rho_1'$ and $\gamma = \gamma_1$ we have

$$\begin{aligned}
\gamma\beta + \rho &= \beta\left(\lfloor a_1 \rceil + i\sqrt{3}\left\lfloor b_1/\sqrt{3}\right\rceil\right) + \beta\left((a_1 - \lfloor a_1 \rceil) + i\left(b_1 - \sqrt{3}\left\lfloor b_1/\sqrt{3}\right\rceil\right)\right) \\
&= \beta\left(a_1 + b_1 i\right) \\
&= \alpha
\end{aligned}$$

since $(a_1 + b_1 i) = \beta^{-1}\alpha$. Similarly one can also check the case where we set $\rho = \rho_2 = \beta\rho_2'$ and $\gamma = \gamma_2$.

Next we show that $\gamma\beta$ is a element of $\mathcal{L}(\beta)$ closest to α. In Section 2.2 we saw the closest lattice point on a rectangular lattice can easily be found if we work in coordinates relative to the orthogonal basis. We can transform the lattice generated by $\{\beta, \beta\omega\}$ into a lattice generated by $\{1, \omega\}$ by multiplying by β^{-1}. Then we can find the closest vector to $\beta^{-1}\alpha$ on the lattice generated by $\{1, \omega\}$ which is the set of Eisenstein integers, as below.

Recall the Eisenstein integers can be partitioned into a sublattice $\mathcal{L}_R$ and its coset $\mathcal{L}_R + \omega$. The vector ρ_1' is the vector to a closest lattice point γ_1' in $\mathcal{L}_R$ to $v_1 = a_1 + b_1 i$. The vector ρ_2' determines the vector to a closest lattice point γ_2' in $\mathcal{L}_R$ to $v_2 = a_2 + b_2 i$ which is equivalent to finding the vector to the closest lattice point in $\mathcal{L}_R + \omega$ to $v_1 = a_1 + b_1 i$. It follows that $d(\rho)$ is the minimum element of $\{d(x) | x \in \alpha + \mathcal{L}(\beta)\}$ or equivalently ρ is a smallest representative of the congruence class modulo β. It also clear that $d(\rho) < d(\beta)$.

We now show that the division algorithm always outputs a unique representative of the congruence class, that is if $\alpha, \alpha' \in \mathbb{Z}[\omega]$ satisfy $\alpha' \equiv \alpha \mod \beta$ then the algorithm produces the same output when applied to (α, β) and (α', β). As we have shown the algorithm is a closest vector algorithm, we only need to consider the cases where α is equidistant to two or more points in $\mathcal{L}(\beta)$. Or equivalently we assume $\beta^{-1}\alpha$ is equidistant to two or more Eisenstein integers.

Since the geometry relative to the lattice of equivalent vectors is the same, always choosing the closest point to $\beta^{-1}\alpha$ furthest to the right (or up), will give the same representative of the congruence class. We show that the division algorithm always chooses the closest point to $\beta^{-1}\alpha$ furthest to the right (or up).

In the case where there are two closest points in $\mathcal{L}_R$ equidistant to $\beta^{-1}\alpha$ the consistency of the nearest integer function (rounding 0.5 up) in ρ_1 guarantees we always choose the closest point in $\mathcal{L}_R$ the furthest to the right (or above) of $\beta^{-1}\alpha$. Similarly

ρ_2 always chooses the closest point to the right (or above) of $\beta^{-1}\alpha$ in $\mathcal{L}_R + \omega$. In the case $d(\rho_1) = d(\rho_2)$ the closest elements to $\beta^{-1}\alpha$ in $\mathcal{L}_R$ and $\mathcal{L}_R + \omega$ are equidistant from $\beta^{-1}\alpha$ we always choose the representative furthest to the right. That is if $Re(\gamma_1) > Re(\gamma_2)$ set $(\rho, \gamma) = (\rho_1, \gamma_1)$ otherwise if $Re(\gamma_1) < Re(\gamma_2)$ set $(\rho, \gamma) = (\rho_2, \gamma_2)$. Since $\mathcal{L}_R$ and $\mathcal{L}_R + \omega$ are offset $Re(\gamma_1) \neq Re(\gamma_2)$. Therefore the division algorithm always outputs a unique representative since we always choose the closest point to $\beta^{-1}\alpha$ furthest to the right (or up). ∎

Note that when $d(\rho_1) = d(\rho_2)$, the algorithm presented in [29] always chose ρ_1 instead of choosing the point furthest to the right. This lead to some occurrences where different representatives of a congruence class were output.

As before we define a **fundamental domain** of $\mathcal{L}(\beta)$ to be a subset of $\mathbb{Z}[\omega]$ that contains a unique representative from each the of the distinct congruence classes modulo β.

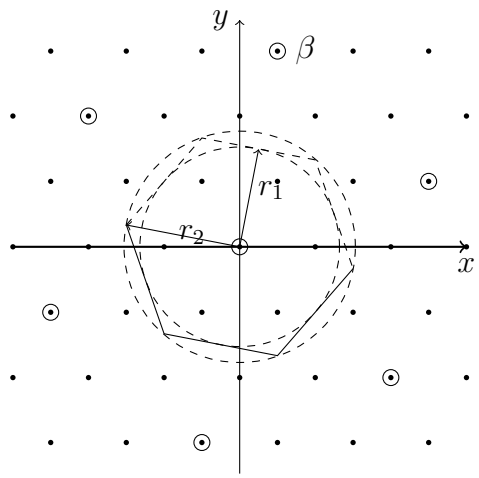

Figure 4.2: The dots represent elements of $\mathbb{Z}[\omega]$. The open circles represent the lattice $\mathcal{L}(\beta)$ with $\beta = 2 + 3\omega$. The points in the hexagon represents a fundamental domain of $\mathcal{L}(\beta)$. The inscribed circle has radius $r_1 = \frac{\sqrt{d(\beta)}}{2}$ and the outer circle has radius $r_1 = \sqrt{\frac{d(\beta)}{3}}$.

For example the division algorithm in Theorem 4.1.1 for reduction modulo β maps the Eisenstein integers to a fundamental domain of $\mathcal{L}(\beta)$. The edges of the hexagon in Figure 4.2 are equidistant from exactly two lattice points in $\mathcal{L}(\beta)$ and the vertices of the hexagon are equidistant from exactly three lattice points. It is clear that the Eisenstein integers in the hexagon in Figure 4.2 are the smallest representatives of the congruences classes modulo β. The points on opposite edges of the hexagon are equivalent modulo β, therefore we only include one of each opposite edge in the fundamental domain. One can apply the division algorithm to see which edges to include.

Notice the division algorithm in Theorem 4.1.1 finds $\rho = \alpha - \gamma\beta$ where $\gamma\beta$ is a closest vector to α in $\mathcal{L}(\beta)$. Since ρ is inside the hexagon given by the fundamental domain illustrated in Figure 4.2 it is clear $d(\rho) < d(\beta)$. Hence we deduce that the Eisenstein integers are an Euclidean domain.

4.2 Properties of Eisenstein Integers

We will now look at properties of Eisenstein integers. Most of the material for this section can be found in [22] however the proofs are my own.

Recall that if R is any ring, then an element $\mu \in R$ is called a **unit** if μ has a multiplicative inverse in R.

Proposition 4.2.1 *An Eisenstein integer $\mu \in \mathbb{Z}[\omega]$ is a unit if and only if $d(\mu) = 1$.*

Proof: If $d(\mu) = 1$ then we have $\mu\bar{\mu} = 1$. Since $\bar{\mu} \in \mathbb{Z}[\omega]$, this implies that $\bar{\mu}$ is the multiplicative inverse of μ, hence μ is a unit.

Now suppose that $\mu \in \mathbb{Z}[\omega]$ is a unit. Then there exists a non-zero $\alpha \in \mathbb{Z}[\omega]$ such that $\mu\alpha = 1$. Hence $d(\mu)\,d(\alpha) = d(\mu\alpha) = 1$. Since $d(\mu)$ and $d(\alpha)$ are positive integers this implies that $d(\mu) = 1$. ∎

It is clear that the elements $\pm 1, \pm\omega, \pm\omega^2 \in \mathbb{Z}[\omega]$ are units since $\omega^3 = 1$ hence they have multiplicative inverses. The following proposition shows that they are the only units of $\mathbb{Z}[\omega]$.

Proposition 4.2.2 *The units of* $\mathbb{Z}[\omega]$ *are* ± 1, $\pm\omega$ *and* $\pm\omega^2$.

Proof: If $\mu = a + b\omega \in \mathbb{Z}[\omega]$, with $a, b \in \mathbb{Z}$, is a unit then we know by Proposition 4.2.1 that $d(\mu) = 1$ hence

$$a^2 + b^2 - ab = 1. \tag{4.2.1}$$

It follows that $a^2 + b^2 - 2ab + ab = 1$ and therefore

$$(a - b)^2 = 1 - ab.$$

Since $a^2 + b^2 \geq 0$ we have $1 + ab \geq 0$ hence $-1 \leq ab$ and since $(a - b)^2 \geq 0$ we have $1 - ab \geq 0$ hence $ab \leq 1$. Therefore $-1 \leq ab \leq 1$ so $a, b \in \{-1, 0, 1\}$.

Equation (4.2.1) does not hold if both a and b are 0 or if a and b have opposite signs, that is $a = -1$ and $b = 1$ or $a = 1$ and $b = -1$.

This leaves the cases where $a = 0$ and $b = \pm 1$ or $b = 0$ and $a = \pm 1$ or $a = 1$ and $b = 1$ or $a = -1$ and $b = -1$. Equation (4.2.1) holds for each of these six cases. Therefore the units of $\mathbb{Z}[\omega]$ are ± 1, $\pm\omega$, $\omega^2 = -1 - \omega$ and $-\omega^2 = 1 + \omega$. ∎

We now look at the prime elements of $\mathbb{Z}[\omega]$. We begin by presenting common definitions and theorems that can be found in any standard abstract algebra textbook such as [5]. Recall that a nonzero element p of a ring $\mathcal{R}$ is **prime** if it is not a unit and if $p|ab$ for any $a, b \in \mathcal{R}$ then $p|a$ or $p|b$. Also recall that a nonzero element p of a ring $\mathcal{R}$ is **irreducible** if p is not a unit and if $p = ab$ with $a, b \in \mathcal{R}$ then either a or b is a unit. Two elements a and b in a ring $\mathcal{R}$ are called **associates** if $a = \mu b$ for some unit $\mu \in \mathcal{R}$. Note that if p is prime then associates of p are also prime.

Note that in a general integral domain, each prime element is irreducible, but the converse is not necessarily true. However the ring of Eisenstein integers is an Euclidean Domain. Every Euclidean Domain is a Principal Ideal Domain (PID), an integral domain in which every ideal is generated by a single element, and in a PID an element is prime if and only if it is irreducible.

Also recall that every PID is a Unique Factorization Domain (UFD). This means that any nonzero Eisenstein integer which is not a unit can be written as a product of irreducibles and this factorization is unique up to associates.

Note that primes in $\mathbb{Z}$ are not necessarily prime in $\mathbb{Z}[\omega]$. For example 13 is a prime integer, but 13 can be written as a product of two Eisenstein integers: $13 = (4+\omega)(3-\omega)$ neither of which is a unit. We will distinguish between primes in $\mathbb{Z}[\omega]$ and primes in $\mathbb{Z}$ by referring to primes in $\mathbb{Z}[\omega]$ as *Eisenstein primes* and primes in $\mathbb{Z}$ as *rational primes*.

Our goal for the remainder of this section is to classify all the Eisenstein primes.

Proposition 4.2.3 *If ϕ is an Eisenstein prime then there exists a rational prime p such that $d(\phi) \in \{p, p^2\}$. In the case where $d(\phi) = p$ then ϕ is not associate to a rational prime. In the case where $d(\phi) = p^2$ then ϕ is associate to the rational prime p.*

Proof: Let ϕ be an Eisenstein prime. Then $d(\phi) = \phi\bar{\phi} = n$ for some $n \in \mathbb{Z}$ and since ϕ is not a unit we have $n > 1$. Since $n > 1$ we can write n as a product of rational primes. Let $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$. Since ϕ is prime this means that ϕ divides a rational prime p_i for some $1 \leq i \leq k$.

Suppose $p_i = \phi\alpha$ for some $\alpha \in \mathbb{Z}[\omega]$. Then $d(\phi)\, d(\alpha) = d(\phi\alpha) = d(p_i) = p_i^2$. Therefore either $d(\phi) = p_i^2$ and $d(\alpha) = 1$ or $d(\phi) = p_i$ and $d(\alpha) = p_i$. Note that $d(\phi) \neq 1$ since ϕ is not a unit.

In the case where $d(\phi) = p_i^2$ and $d(\alpha) = 1$ we have that α is a unit and thus has an inverse α^{-1} which is also a unit. Hence $\phi = \alpha^{-1} p_i$ which means ϕ is associate to

a rational prime.

Now consider the case where $d(\phi) = p_i$ and $d(\alpha) = p_i$. Suppose to the contrary that ϕ is associate to a rational prime q. Then $\phi = \mu q$ for some unit $\mu \in \mathbb{Z}[\omega]$. Then $p_i = d(\phi) = d(\mu)\, d(q) = q^2$ which is a contradiction since p_i is prime. Hence in the case where $d(\phi) = p_i$, ϕ is not associate to a rational prime. ∎

The following proposition gives a partial converse to Proposition 4.2.3:

Proposition 4.2.4 *If $\phi \in \mathbb{Z}[\omega]$ and $d(\phi) = p$ where p is a rational prime, then ϕ is a prime in $\mathbb{Z}[\omega]$.*

Proof: We prove the contrapositive. Suppose that ϕ is not an Eisenstein prime. We can write $\phi = \alpha\beta$ for some $\alpha, \beta \in \mathbb{Z}[\omega]$ with $d(\alpha) > 1$ and $d(\beta) > 1$, that is α and β are not units. Then $p = d(\phi) = d(\alpha)\, d(\beta)$ which shows that $d(\phi)$ is not a rational prime. ∎

Proposition 4.2.5 *Suppose that ϕ is an Eisenstein integer. Then $d(\phi) = 3$ if and only if $\phi = \mu(1 - \omega)$ for some unit μ.*

Proof: We denote $\pi = 1 - \omega$.

If $\phi = \mu\pi$ for some unit μ, then $d(\phi) = d(\mu)d(\pi) = 3$.

Now suppose that $d(\phi) = 3$ and let us prove that $\phi = \mu\pi$. Let $\phi = a + b\omega$ then

$$a^2 + b^2 - ab = 3. \tag{4.2.2}$$

It follows that $a^2 + b^2 - 2ab + ab = 3$ and therefore

$$(a - b)^2 = 3 - ab.$$

Since $a^2 + b^2 \geq 0$ we have $3 + ab \geq 0$ hence $-ab \leq 3$ and since $(a-b)^2 \geq 0$ we have $3 - ab \geq 0$ hence $ab \leq 3$ and therefore $-3 \leq ab \leq 3$. Either a and b have opposite signs, or a and b have the same sign, or either a or b are zero.

Let us consider the case where a and b have opposite signs. Suppose $a < 0$ and $b > 0$. The only integer solutions to $ab \geq -3$ are $a = -3$ and $b = 1$ or $a = -1$ and $b \in \{1, 2, 3\}$. Of these only $a = -1$ and $b = 1$ satisfy Equation (4.2.2) giving $\phi = -1 + \omega = (-1)\pi$. By symmetry the only solution to $ab \geq -3$ with $a > 0$ and $b < 0$ that satisfies Equation (4.2.2) is $a = 1$ and $b = -1$ yielding $\phi = 1 - \omega = (1)\pi$.

Now let us consider the case where a and b have the same sign. Suppose $a > 0$ and $b > 0$. The only integer solutions that satisfy $ab \leq 3$ are $a = 1$ and $b \in \{1, 2, 3\}$ or $b = 1$ and $a \in \{1, 2, 3\}$. Of these only $\phi = 2 + \omega = -\omega^2\pi$ and $\phi = 1 + 2\omega = \omega\pi$ satisfy equation 4.2.2. By symmetry, since $d(\phi) = d(-\phi)$, we have that $\phi = -2 - \omega = \omega^2\pi$ and $\phi = -1 - 2\omega = -\omega\pi$ are the only solutions for $a < 0$ and $b < 0$.

Finally let us consider the case where either a or b are zero. If $a = 0$ then by Equation (4.2.2) we have $b^2 = 3$ which does not have an integer solution. Similarly if $b = 0$ then by Equation (4.2.2) we have $a^2 = 3$ which also does not have an integer solution.

Therefore the only solutions to $d(\phi) = 3$ are $\phi = (\pm 1)\pi$, $\phi = \pm\omega\pi$ and $\phi = \pm\omega^2\pi$. Hence $d(\phi) = 3$ if and only if $\phi = \mu(1 - \omega)$ for some unit μ. ∎

Notice that Propositions 4.2.4 and 4.2.5 show that $1 - \omega$ and its associates are all the primes with $d(\phi) = 3$. The following theorem given in [22] gives three families of Eisenstein primes. We expand upon this and show in Theorem 4.2.7 that all the Eisenstein primes belong to these three families. We will need the relation $\sqrt{-3} = 1 + 2\omega$ which follows from $\omega = \frac{1}{2}(-1 + \sqrt{3}i)$.

Theorem 4.2.6 *[22, Proposition 9.1.4] Suppose that p is a rational prime. Then the following is true:*

1. $1-\omega$ *and its associates are Eisenstein primes.*

2. *If* $p \equiv 2 \mod 3$*, then* p *and its associates are Eisenstein primes.*

3. *If* $p \equiv 1 \mod 3$*, then* $p = d(\phi)$ *for some Eisenstein prime* ϕ*.*

Proof:

Since $d(1-\omega) = 3$ and 3 is a rational prime it is clear by Proposition 4.2.4 that $1-\omega$ and its associates are Eisenstein primes. Therefore case (1) holds.

Let $p \neq 3$ be a rational prime. We will now show that p is an Eisenstein prime if and only if $p \equiv 2 \mod 3$.

If p is not an Eisenstein prime, then $p = \alpha\beta$ for some $\alpha, \beta \in \mathbb{Z}[\omega]$ with $d(\alpha) > 1$ and $d(\beta) > 1$. This means that $p^2 = d(p) = d(\alpha)\, d(\beta)$. Hence $d(\alpha) = p$ and $d(\beta) = p$.

Let $\alpha = a + b\omega$ with $a, b \in \mathbb{Z}$ then $p = d(\alpha) = a^2 + b^2 - ab$. We can rewrite this as

$$(a+b)^2 - 3ab = p.$$

Therefore $p \equiv (a+b)^2 \mod 3$ and it follows that $p \equiv 1 \mod 3$ since 3 does not divide p. Hence if p is not an Eisenstein prime then $p \equiv 1 \mod 3$.

Recall that x is a quadratic residue modulo y if there exists an integer z such that $z^2 \equiv x \mod y$. Also recall the law of quadratic reciprocity which states that if $q \equiv 1 \mod 4$ then q is a quadratic residue modulo p if and only if p is a quadratic residue modulo q [22, Chapter 2, Theorem 2].

Now suppose that $p \equiv 1 \mod 3$ so p is a quadratic residue modulo 3. It follows that $p = 1 + y(3)$ for some $y \in \mathbb{Z}$ hence $p = 1 - y(-3)$ and $p \equiv 1 \mod (-3)$ so p is also a quadratic residue modulo -3. Consequently, by the law of quadratic reciprocity taking $q = -3 \equiv 1 \mod 4$, we deduce that -3 is a quadratic residue modulo p. This means that there exists an $a \in \mathbb{Z}$ such that $a^2 \equiv -3 \mod p$ or $a^2 = -3 + bp$ for some $b \in \mathbb{Z}$. Recall that $\sqrt{-3} = 1 + 2\omega$, hence $a^2 + 3 = (a - \sqrt{-3})(a + \sqrt{-3}) =$

$(a-1-2\omega)(a+1+2\omega)$. We have that p divides a^2+3, so if p is an Eisenstein prime then p divides one of the factors of a^2+3. For $p \in \mathbb{Z}$ and $p|c+d\omega$ we have $p|c$ and $p|d$. This implies that p must divide 2 since p is an integer. However p is a rational prime and $p \neq 2$ since $p \equiv 1 \mod 3$, therefore p does not divide 2, hence p does not divide one of the factors of a^2+3 which means that p is not an Eisenstein prime.

Therefore a rational prime $p \neq 3$ is not an Eisenstein prime if and only if $p \equiv 1 \mod 3$. If follows that $p \neq 3$ is an Eisenstein prime if and only if $p \equiv 2 \mod 3$. Hence case (2) holds.

Finally, if $p \equiv 1 \mod 3$ then p is not an Eisenstein prime and $p = \phi\alpha$ where ϕ and α are not units. Hence $d(p) = d(\phi)\,d(\alpha)$ so $p^2 = d(\phi)\,d(\alpha)$. It follows that $d(\phi) = p$ which means ϕ is prime by Proposition 4.2.4. Hence case (3) holds. ∎

Theorem 4.2.7 *If ϕ is an Eisenstein prime then exactly one of the following conditions is satisfied:*

1. *ϕ is associated to $1-\omega$.*
2. *ϕ is associated to a rational prime p satisfying $p \equiv 2 \mod 3$.*
3. *$d(\phi)$ is a rational prime p satisfying $p \equiv 1 \mod 3$.*

Proof: Let ϕ be an Eisenstein prime. Proposition 4.2.3 states that if ϕ is an Eisenstein prime then $d(\phi) = p$ or $d(\phi) = p^2$ for some rational prime p.

Suppose that $d(\phi) = p$ for some rational prime p. We have that either $p = 3$ or $p \equiv 1 \mod 3$ or $p \equiv 2 \mod 3$. If $p = 3$ then Proposition 4.2.5 states that ϕ is a prime associate to $1-\omega$. Hence ϕ is in category (1). If $p \equiv 2 \mod 3$ then Theorem 4.2.6 implies that p is an Eisenstein prime. However $p = d(\phi) = \phi\bar{\phi}$. This is a contradiction hence $p \equiv 1 \mod 3$ and falls into category (3).

Now suppose that $d(\phi) = p^2$ for some rational prime p. By Proposition 4.2.3 we have that ϕ is associate to p. This means that p is an Eisenstein prime. Note that $p = 3$ is not an Eisenstein prime since $\sqrt{-3} = 1 + 2\omega$ hence $3 = -(1 + 2\omega)^2$ and $(1 + 2\omega)$ is not a unit. We know that $p \neq 3$ is an Eisenstein prime if and only if $p \equiv 2 \mod 3$. Hence if $d(\phi) = p^2$, ϕ falls into category (2).

Therefore all the Eisenstein primes can be classified into one of the three categories of Theorem 4.2.6. ∎

Proposition 4.2.8 *Let ϕ be an Eisenstein prime. Then there are either six or twelve Eisenstein primes with the same algebraic norm.*

Proof: Let ϕ be an Eisenstein prime. Then ϕ is in one of the three categories of Theorem 4.2.7.

Suppose ϕ is in category (1). Then $d(\phi) = 3$ and it follows from Proposition 4.2.5 that only the six associates of ϕ have algebraic norm 3.

Suppose ϕ is in category (2). Then ϕ is associate to a rational prime $p \equiv 2 \mod 3$ and $d(\phi) = p^2$. It follows from Proposition 4.2.3 that if $\psi \in \mathbb{Z}[\omega]$ satisfies $d(\psi) = p^2$ where p is a rational prime, then ψ is associate to p and hence to ϕ. So the six associates of p are the only Eisenstein primes with algebraic norm p^2.

Suppose ϕ is in category (3). Then $d(\phi) = p$ where p is a rational prime and $p \equiv 1 \mod 3$. It follows that $\phi\bar{\phi} = d(\phi) = p$. Since the ring of Eisenstein integers is a UFD this factorization is unique up to associates. Hence the associates of ϕ and $\bar{\phi}$ are the only Eisenstein primes with algebraic norm p. Therefore in the case where $\bar{\phi}$ is not an associate of ϕ there are twelve Eisenstein primes with algebraic norm p, and otherwise there are six Eisenstein primes with algebraic norm p. ∎

4.3 Size of $\mathbb{Z}[\omega]/(\alpha)$

It is important to know how much storage is needed to store polynomials with coefficients modulo Eisenstein integers. In this section we count the number of distinct congruence classes modulo an Eisenstein integer α. This is equivalent to determining the number of points of the lattice $\mathbb{Z}[\omega]$ that lie in the fundamental domain of $\mathcal{L}(\alpha)$ illustrated in Figure 4.2 for $\beta = \alpha$. We extend [22, Proposition 9.2.1] to count the number of congruence classes for any Eisenstein integer. We begin with two lemmas.

Lemma 4.3.1 *Let $\alpha = a+b\omega$, with $a, b \in \mathbb{Z}$ be an Eisenstein integer. If $d = \gcd(a, b)$ then $d^2 | d(\alpha)$.*

Proof: Let $d = \gcd(a, b)$. Then $a = zd$ and $b = yd$ for some integers z and y with $\gcd(z, y) = 1$. We have that $d(\alpha) = a^2 + b^2 - ab$ so $d(\alpha) = z^2d^2 + y^2d^2 - zyd^2$ hence

$$d(\alpha) = d^2\left(z^2 + y^2 - zy\right)$$

therefore d^2 divides $d(\alpha)$. ∎

Lemma 4.3.2 *Let $\alpha = a + b\omega$ be a nonzero Eisenstein integer (where $a, b \in \mathbb{Z}$). If $d = \gcd(a, b)$ then $\frac{d(\alpha)}{d} \in \mathcal{L}(\alpha)$ and $\frac{d(\alpha)}{d}$ is the smallest positive integer in $\mathcal{L}(\alpha)$.*

Proof: Let $\alpha = a + b\omega$ then $\bar{\alpha} = a + b\bar{\omega} = a + b(-1 - \omega) = (a - b) - b\omega$.

If $d = \gcd(a, b)$ then $d|a$ and $d|b$. It follows that $d|(a - b)$ hence $\frac{\bar{\alpha}}{d}$ is also an Eisenstein integer. Therefore $\frac{d(\alpha)}{d} = \alpha\frac{\bar{\alpha}}{d} \in \mathcal{L}(\alpha)$.

Suppose $\lambda\bar{\alpha} = \lambda(a - b - b\omega) \in \mathbb{Z}[\omega]$ with $\lambda \in \mathbb{R}$ and $\lambda \neq 0$. It follows that $\lambda(a - b) \in \mathbb{Z}$ and $\lambda b \in \mathbb{Z}$. Therefore λ must be a rational number of the form $\lambda = \frac{m}{n}$ for some relatively prime $m, n \in \mathbb{Z}$ with $n|(a - b)$ and $n|b$. Let $y \in L(\alpha) \cap \mathbb{Z}$. Then $y = \lambda\bar{\alpha}\alpha$ with $\lambda\bar{\alpha} \in \mathbb{Z}[\omega]$ for some $\lambda \in \mathbb{R}$. Since $d = \gcd(a, b) = \gcd(a - b, b)$, d is the largest integer that divides $\bar{\alpha}$. Therefore $\lambda = \frac{1}{d}$ is the smallest positive value λ

can take. Hence $y = \frac{1}{d}\bar{\alpha}\alpha = \frac{d(\alpha)}{d}$ is the smallest positive integer in $\mathcal{L}(\alpha)$. ∎

Theorem 4.3.3 *Let α be any nonzero Eisenstein integer. There are exactly $d(\alpha)$ elements in $\mathbb{Z}[\omega]/(\alpha)$, that is there are exactly $d(\alpha)$ congruence classes modulo α.*

Proof: Let $\alpha = a+b\omega$, with $a, b \in \mathbb{Z}$, be an Eisenstein integer and let $d = \gcd(a, b)$. By Lemma 4.3.2, $x = \frac{d(\alpha)}{d} \in \mathbb{Z}$. We will show that

$$\{u + v\omega | u, v \in \mathbb{Z}, 0 \leq u < x, 0 \leq v < d\}$$

is a complete set of representatives for the congruence classes modulo α. This will show that there are exactly $xd = d(\alpha)$ congruence classes.

Let $\beta = m + n\omega$, with $m, n \in \mathbb{Z}$, be any Eisenstein integer. Then $n \equiv v \mod d$ with $0 \leq v < d$. Hence we can write $n = v + kd$ for some integer k. Since $d = \gcd(a, b)$ we know there exist integers s and t such that $sb + ta = d$. It follows that $ksb + kta = kd$ hence

$$n - ksb - kta = n - kd = v.$$

We have that $\beta - \gamma \equiv \beta \mod \alpha$ for any $\gamma \in \mathcal{L}(\alpha)$. Our goal is to find $\gamma \in \mathcal{L}(\alpha)$ such that $\beta - \gamma = u + v\omega$ with $0 \leq u < x$ and $0 \leq v < d$. We have that

$$\begin{aligned} \beta - ks\alpha + kt\omega^2\alpha &= (m + n\omega) - ks(a + b\omega) + kt(-a + b - a\omega) \\ &= (m - ksa - kta + ktb) + (n - ksb - kta)\omega \\ &= (m - ksa - kta + ktb) + v\omega. \end{aligned}$$

We can write $m - ksa - kta + ktb \equiv u \mod x$ with $0 \leq u < x$, therefore we have

$$m - ksa - kta + ktb = u + lx \tag{4.3.1}$$

for some integer l.

Since α, $\omega^2\alpha$ and x are in $\mathcal{L}(\alpha)$ we have $\beta - ks\alpha + kt\omega^2\alpha - lx \equiv \beta \mod \alpha$. Therefore

$$\begin{aligned}\beta - ks\alpha + kt\omega^2\alpha - lx &= (m - ksa - kta + ktb - lx) + (n - ksb - kta)\,\omega \\ &= u + v\omega\end{aligned}$$

with $0 \leq u < x$ and $0 \leq v < d$.

We have shown that every Eisenstein integer is congruent modulo α to an Eisenstein integer of the form $u + v\omega$ with $0 \leq u < x$ and $0 \leq v < d$. Now we need to show that this is a unique set of congruence classes. Let $\beta = m + n\omega$ and $\beta' = m' + n'\omega$ with $0 \leq m, m' < x$ and $0 \leq n, n' < d$. Suppose that $\beta \equiv \beta' \mod \alpha$. We will show that $m = m'$ and $n = n'$.

Since $\beta \equiv \beta' \mod \alpha$, $\beta - \beta' = \alpha\gamma$ for some $\gamma \in \mathbb{Z}[\omega]$. Since $\alpha = a + b\omega$ and $d = \gcd(a, b)$, we know that $\frac{\alpha}{d} \in \mathbb{Z}[\omega]$. Therefore

$$\frac{(m - m') + (n - n')\,\omega}{d} = \frac{\beta - \beta'}{d} = \frac{\alpha}{d}\gamma \in \mathbb{Z}[\omega]$$

hence $d|\,(m - m')$ and $d|\,(n - n')$. It follows that $n - n' = 0$ since $0 \leq n - n' < d$ so $n = n'$. Hence we have that $(m - m') = \alpha\gamma$ consequently $\alpha\gamma$ is an integer. We know that x is the smallest non zero integer in $\mathcal{L}(\alpha)$ and since $0 \leq m - m' < x$ it follows that $m - m' = 0$.

Hence $\{u + v\omega | u, v \in \mathbb{Z}, 0 \leq u < x, 0 \leq v < d\}$ is a complete set of representatives for the congruence classes modulo α and there are exactly $xd = d(\alpha)$ congruence classes. ∎

We have shown in the proof of Theorem 4.3.3 that the parallelogram represented by $\{u + v\omega | u, v \in \mathbb{Z}, 0 \leq u < x, 0 \leq v < d\}$ is a complete set of representatives for the congruence classes modulo α in $\mathbb{C}$ and hence the intersection of this parallelogram with $\mathbb{Z}[\omega]$ is another *fundamental domain* for $\mathcal{L}(\alpha)$, as illustrated in Figure 4.3.

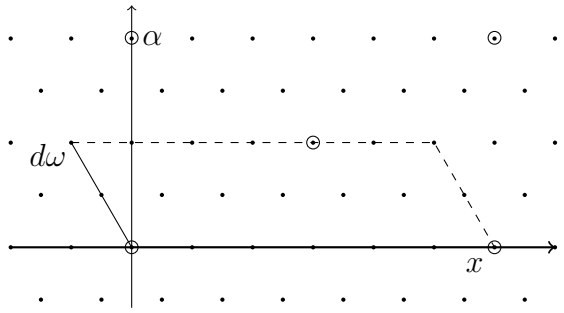

Figure 4.3: The dots are $\mathbb{Z}[\omega]$. The open circles represent the lattice $\mathcal{L}(\alpha)$ with $\alpha = 2 + 4\omega$. We have $d = \gcd(2, 4) = 2$ and $x = \frac{d(\alpha)}{d} = 6$. The parallelogram with vertices 0, x, $d\omega$ and $x + d\omega$ represents another fundamental domain of $\mathcal{L}(\alpha)$.

Notice in the case where $d = 1$, that is in the case for $\alpha = a + b\omega$ where the $\gcd(a, b) = 1$ then the set of distinct congruence classes modulo α can be represented by the set $\{0, 1, \ldots, d(\alpha) - 1\}$. For example for $\alpha = 2 + 3\omega$ we have $d(\alpha) = 7$ and hence the set $\{0, 1, 2, 3, 4, 5, 6\}$ is a set of distinct congruence classes modulo α.

4.4 Implementing ETRU

We have found a division algorithm for the Eisenstein integers that outputs a smallest and unique representative of a congruence class. Therefore we have all we need to extend NTRU over the Eisenstein integers (ETRU).

We set

$$\mathcal{R} = \mathbb{Z}[\omega][x]/\left\langle x^N - 1 \right\rangle$$

which is the set of convolution polynomials of degree $N - 1$ and we need p and q to be relatively prime. As before we let $\mathcal{L}_f$,$\mathcal{L}_g$,$\mathcal{L}_\phi$ and $\mathcal{L}_m$ be subsets of $\mathcal{R}$.

The encryption and decryption algorithms are the same as in Section 3.1.1. The inverses of polynomials modulo p and q can be found using the extended Euclidean algorithm.

4.4.1 Selecting p and q

As in the case with classical NTRU we need to be able to easily calculate the inverse of f modulo p and q thus we want to choose p and q to be prime or a prime power.

We choose p to be $2 + 3\omega$ because $d(p) = 7$ and the distinct equivalence classes modulo p are 0 and the six units $\pm 1, \pm\omega, \pm\omega^2$, see Figure 4.2. We will choose q to be prime or a power of $1 - \omega$ the smallest prime. Note that this is not the only possible choice for p, for example we could take p to be another small prime such as $1 - \omega$.

4.4.2 The Forms of f, g and ϕ

We will choose $f \in \mathcal{L}_f$, $g \in \mathcal{L}_g$ and $\phi \in \mathcal{L}_\phi$ to be polynomials with coefficients from $\{0, \pm 1, \pm\omega, \pm\omega^2\}$. The polynomials g and ϕ will respectively have d_g and d_ϕ of each of the units $\pm 1, \pm\omega$ and $\pm\omega^2$ and the rest of their coefficients zero. Similar to classical NTRU we will choose f to have $d_f + 1$ 1's and d_f of each of the other units so that $x - 1$ is not a factor of f.

4.4.3 Implementation

We implemented the ETRU encryption and decryption algorithms with C++ using the same algorithms described for NTRU in Section 3.1 replacing polynomials with integer coefficients with polynomials in $\mathcal{R} = \mathbb{Z}[\omega][x]/\left\langle x^N - 1 \right\rangle$. We also implemented the division algorithm for Eisenstein integers presented by Theorem 4.1.1. The code for the division algorithm is included in Appendix A, as a sample.

The functions for multiplying, dividing and finding inverses (etc...) of polynomials are the same as before, however we replace operations with integers to operations with Eisenstein integers.

As before all tests are run on a computer with a 2.33GHz Intel Core2 Quad Q8200 processor using the Window Vista(64-bit) operating system.

4.4.4 Decryption Failure

We need to be able to choose parameters so that there is a low chance of decryption failure. In the decryption process we compute

$$a \equiv f * e \equiv p\phi * g + f * m \mod q.$$

A decryption failure occurs if the polynomial $\mathrm{A} = p\phi * g + f * m$ changes when we reduce its coefficients modulo q. Therefore we want to be able to compute the probability that all the coefficients of A lie in the fundamental domain of q given by the division algorithm. We will be modifying Kouzmenko's approach in [25] for NTRU over the Gaussian integers to develop a model for the probability of decryption failure.

We define $L(d) = \{f | f \in R$ with d coefficients equal to each of 1, -1, ω, $-\omega$, ω^2, $-\omega^2$ and the rest $0\}$. Let d_f, d_g, d_ϕ and d_m be positive integers. To simplify our presentation we will assume $\mathcal{L}_f = L(d_f)$, $\mathcal{L}_g = L(d_g)$, $\mathcal{L}_\phi = L(d_\phi)$ and $\mathcal{L}_m = L(d_m)$.

Let us write $i + j \equiv k$ for $i + j \equiv k \mod N$. We know that the jth coefficient of A is given by

$$\mathrm{A}_j = p \sum_{k+l \equiv j} \phi_k g_l + \sum_{k+l \equiv j} f_k m_l.$$

Let $\alpha = a + b\omega = a + \frac{b}{2}\left(-1 + \sqrt{3}i\right)$ be an Eisenstein integer. Then $Re(\alpha) = a - \frac{b}{2}$ is the real part of α and $Im(\alpha) = \frac{\sqrt{3}}{2}b$ is the imaginary part of α.

We have that

$$\sum_{k+l \equiv j} \phi_k g_l = \sum_{k+l \equiv j} Re(\phi_k g_l) + \sum_{k+l \equiv j} Im(\phi_k g_l)\, i.$$

Since we know that $(c_1 + d_1 i)(c_2 + d_2 i) = (c_1 c_2 - d_1 d_2) + (c_1 d_2 + d_1 c_2)\, i$, it follows that

$$p \sum_{k+l \equiv j} \phi_k g_l = \left(Re(p) \sum_{k+l \equiv j} Re(\phi_k g_l) - Im(p) \sum_{k+l \equiv j} Im(\phi_k g_l) \right) + \left(Re(p) \sum_{k+l \equiv j} Im(\phi_k g_l) + Im(p) \sum_{k+l \equiv j} Re(\phi_k g_l) \right) i$$

hence

$$\mathrm{A}_j = \left(Re\,(p) \sum_{k+l\equiv j} Re\,(\phi_k g_l) - Im\,(p) \sum_{k+l\equiv j} Im\,(\phi_k g_l) + \sum_{k+l\equiv j} Re\,(f_k m_l) \right) \tag{4.4.1}$$

$$+ \left(Re\,(p) \sum_{k+l\equiv j} Im\,(\phi_k g_l) + Im\,(p) \sum_{k+l\equiv j} Re\,(\phi_k g_l) + \sum_{k+l\equiv j} Im\,(f_k m_l) \right) i$$

If N is large we can assume by the central limit theorem that $Re\,(\mathrm{A}_j)$ and $Im\,(\mathrm{A}_j)$ are normally distributed. We further assume that $Re\,(\mathrm{A}_j)$ and $Im\,(\mathrm{A}_j)$ are independent random variables. We will examine the validity of these assumptions at the end of this section.

Let $b \in \{1, -1, \omega, -\omega, \omega^2, -\omega^2\}$. Then

$$Prob\,(\phi_k = b) = \frac{d_\phi}{N}, \quad Prob\,(g_l = b) = \frac{d_g}{N}.$$

Since the coefficients of ϕ and g are in $\{0, 1, -1, \omega, -\omega, \omega^2, -\omega^2\}$ we know that $\phi_k g_l \in \{0, 1, -1, \omega, -\omega, \omega^2, -\omega^2\}$ hence $Re\,(\phi_k g_l)$ is either 1, -1, $\frac{1}{2}$, $-\frac{1}{2}$ or 0. Counting the outcomes we have that

$$Prob\,(Re\,(\phi_k g_l) = 1) = Prob\,(Re\,(\phi_k g_l) = -1) = \frac{6 d_\phi d_g}{N^2}$$

and that

$$Prob\left(Re\,(\phi_k g_l) = \frac{1}{2}\right) = Prob\left(Re\,(\phi_k g_l) = -\frac{1}{2}\right) = \frac{12 d_\phi d_g}{N^2}$$

therefore the expected value of $Re\,(\phi_k g_l)$ is 0 and

$$\begin{aligned} Var\,(Re\,(\phi_k g_l)) &= E\left(Re\,(\phi_k g_l)^2\right) - E\,(Re\,(\phi_k g_l))^2 \\ &= \frac{6 d_\phi d_g}{N^2}\,(1)^2 + \frac{6 d_\phi d_g}{N^2}\,(-1)^2 + \frac{12 d_\phi d_g}{N^2}\left(\frac{1}{2}\right)^2 + \frac{12 d_\phi d_g}{N^2}\left(-\frac{1}{2}\right)^2 \\ &= \frac{18 d_\phi d_g}{N^2}. \end{aligned}$$

Similarly since $\phi_k g_l \in \{0, 1, -1, \omega, -\omega, \omega^2, -\omega^2\}$ hence $Im\,(\phi_k g_l)$ is either $\frac{\sqrt{3}}{2}$, $-\frac{\sqrt{3}}{2}$ or 0. We have that

$$Prob\left(Im\,(\phi_k g_l) = \frac{\sqrt{3}}{2}\right) = Prob\left(Im\,(\phi_k g_l) = -\frac{\sqrt{3}}{2}\right) = \frac{12 d_\phi d_g}{N^2}$$

therefore the expected value of $Im(\phi_k g_l)$ is 0 and

$$\begin{aligned} Var(Im(\phi_k g_l)) &= \frac{12 d_\phi dg}{N^2}\left(\frac{\sqrt{3}}{2}\right)^2 + \frac{12 d_\phi dg}{N^2}\left(\frac{-\sqrt{3}}{2}\right)^2 \\ &= \frac{18 d_\phi d_g}{N^2}. \end{aligned}$$

Notice that $Var(Re(\phi_k g_l)) = Var(Im(\phi_k g_l))$. Similarly $Var(Re(f_k m_l)) = Var(Im(f_k m_l)) = \frac{18 d_f d_m}{N^2}$.

Recall that $Var(X+Y) = Var(X) + Var(Y)$ for two independent random variables X and Y. Therefore if we assume each of the products $\phi_k g_l$ for all $k+l=j \mod N$ are independent, and similarly each $f_k m_l$ are independent, we have

$$\begin{aligned} Var\left(\sum_{k+l\equiv j} Re(\phi_k g_l)\right) &= Var\left(\sum_{k+l\equiv j} Im(\phi_k g_l)\right) = N\frac{18 d_\phi d_g}{N^2} = \frac{18 d_\phi d_g}{N} \quad \text{and} \\ Var\left(\sum_{k+l\equiv j} Re(f_k m_l)\right) &= Var\left(\sum_{k+l\equiv j} Im(f_k m_l)\right) = N\frac{18 d_f d_m}{N^2} = \frac{18 d_f d_m}{N}. \end{aligned}$$

Also recall that $Var(bX) = b^2 Var(X)$ for $b \in \mathbb{R}$. Hence the variance for $Re(\mathrm{A}_j)$ in equation 4.4.1 is

$$\begin{aligned} \sigma_R^2 &= Re(p)^2 Var\left(\sum_{k+l\equiv j} Re(\phi_k g_l)\right) + Im(p)^2 Var\left(\sum_{k+l\equiv j} Im(\phi_k g_l)\right) \\ &\quad + Var\left(\sum_{k+l\equiv j} Re(f_k m_l)\right) \\ &= Re(p)^2 \frac{18 d_\phi d_g}{N} + Im(p)^2 \frac{18 d_\phi d_g}{N} + \frac{18 d_f d_m}{N} \\ &= \frac{18}{N}\left(d(p) d_\phi d_g + d_f d_m\right). \end{aligned}$$

Similarly we find that the variance of $Im(\mathrm{A}_j)$, denoted by σ_I^2, is equal to σ_R^2.

Now we want to find the probability that the coefficients of $\mathrm{A} = p\phi * g + f * m$ lie in the fundamental domain of q given by the division algorithm. That is we want the probability that the coefficients of A lie in the hexagon H inscribed between the circles of radius $\sqrt{d(q)}/2$ and $\sqrt{d(q)/3}$.

The joint probability distribution function (p.d.f) of two randomly normally distributed variables X and Y is

$$f(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} \exp\left[-\frac{q(x,y)}{2}\right]$$

where

$$q(x,y) = \frac{1}{1-\rho^2}\left[\left(\frac{x-\mu_X}{\sigma_X}\right)^2 - 2\rho\left(\frac{x-\mu_X}{\sigma_X}\right)\left(\frac{y-\mu_Y}{\sigma_Y}\right) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\right]$$

with μ_X and σ_X the mean and standard deviation of X, μ_Y and σ_Y the mean and standard deviation of Y and ρ the correlation coefficient of X and Y. The joint probability distribution function and can be found in any standard probability textbook such as [16].

As we are assuming that $X = Re(\mathrm{A}_j)$ and $Y = Im(\mathrm{A}_j)$ are independent we have $\rho = 0$. We have $\mu_R = \mu_I = 0$ and $\sigma_R = \sigma_I = \sigma = \sqrt{\frac{18}{N}(d(p)\, d_\phi d_g + d_f d_m)}$. Hence the joint p.d.f of $Re(\mathrm{A}_j)$ and $Im(\mathrm{A}_j)$ is

$$f(x,y) = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{x^2+y^2}{2\sigma^2}\right] \tag{4.4.2}$$

The probability that A_j lies in the hexagon H is then

$$Prob(\mathrm{A}_j \in H) = \int\int_H f(x,y)\, dxdy$$

and therefore, assuming the coefficients of A are independent, the probability that all N coefficients of a lie in H is

$$Prob\left(\wedge_{j=0}^{N-1}\mathrm{A}_j \in H\right) = \left(\int\int_H f(x,y)\, dxdy\right)^N.$$

This leads to the following:

Remark 4.4.1 Assuming that all the coefficients of $\mathrm{A} = p\phi * g + f * m$ are independent and that the real and imaginary part of each coefficient A_i is independent, the probability of decryption failure for ETRU is approximately

$$1 - \left(\int\int_H f(x,y)\, dxdy\right)^N$$

where H is a hexagon inscribed between the circles of radius $\sqrt{d(q)}/2$ and $\sqrt{d(q)/3}$ and $f(x,y)$ is defined by equation 4.4.2.

Notice that the probability of decryption failure decreases for larger $d(q)$ and for smaller d_f, d_g and d_ϕ.

Testing the Model

We use experimental data to test the validity of hypotheses on Theorem 4.4.1. We will test

$$Var\left(\sum_{k+l\equiv j} Re\left(\phi_k g_l\right)\right) = Var\left(\sum_{k+l\equiv j} Im\left(\phi_k g_l\right)\right) = N\frac{18 d_\phi d_g}{N^2} = \frac{18 d_\phi d_g}{N}.$$

We made the assumption that the real and imaginary parts of each coefficient in the product $\phi * g$ are independent. We randomly generate 10000 ϕ's and g's with $N = 31$ and $d_\phi = d_g = 3$. Since the mean and variance should be the same for each coefficient, we look at the coefficient of the x^3 term in the product $\phi * g$. We find the mean and variance of the real part are

$$\mu_R \approx -0.0096 \quad \text{and} \quad S_R \approx 5.2935$$

and the mean and variance of the imaginary part are

$$\mu_I \approx 0.0116 \quad \text{and} \quad S_I \approx 5.2939$$

and the correlation coefficient of the real and imaginary part is

$$\rho = \frac{S_{IR}}{S_I S_R} \approx 0.003326.$$

Notice that the means μ_R and μ_I are close to zero as we expected. The correlation coefficient is also close to zero therefore it appears that our assumption that real and imaginary part are independent is valid. The variance for the real and imaginary

parts are approximately equal as expected. We calculated the variance for both the real and imaginary should be

$$\frac{18 d_\phi d_g}{N} \approx 5.226$$

which is slightly lower than our experimental data. We have rerun the experiment for different parameter sets and this always seems to be the case. We conclude that our assumptions are valid.

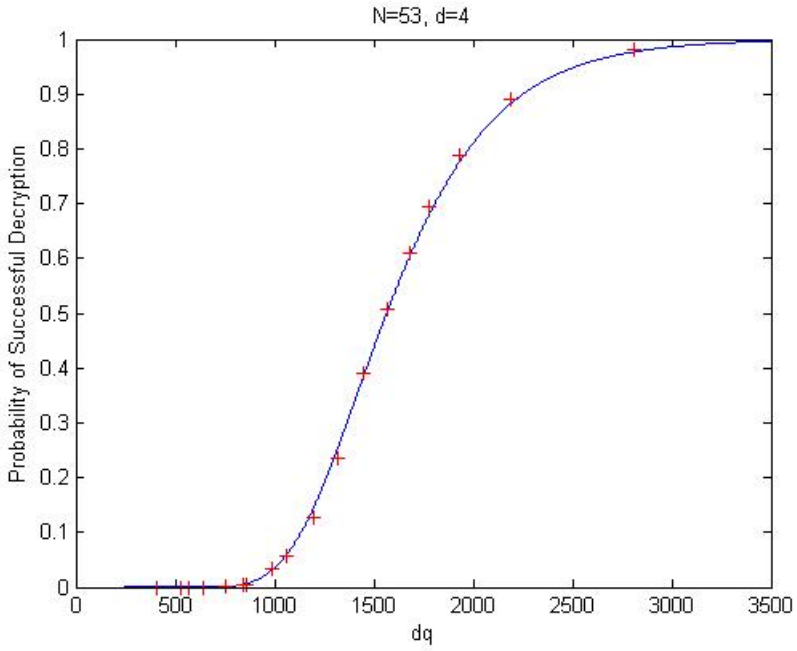

Figure 4.4: Theoretical Model for Decryption Failure

To test our probability model we have coded the ETRU encryption and decryption algorithms using C++. We generate f, g and ϕ as described in Section 4.4.2. The messages m are generated to have random coefficients modulo p.

We have used the parameter set $N = 53$, $d_f = d_g = d_\phi = 4$ and $p = 2 + 3\omega$. We will assume that the coefficients of our messages m are equally distributed hence for our probability model we will take $d_m = \lceil N/7 \rceil = 8$.

The curve in Figure 4.4 represents the theoretical model for the probability that

all the coefficients of the polynomial a lie in the hexagon H described in section 4.4.4. The crosses represent experimental data for different values of q. For each q we encrypt 10000 messages. We then decrypt the encrypted messages and see if they match the original messages. We store the number of messages that decrypted correctly and plot (number of messages decrypted correctly)/(10000). Figure 4.4 shows that the theoretical model given in equation 4.4.1 seems to a good approximation for the probability of decryption success.

4.4.5 Encryption and Decryption Speed

We want to be able to compare the encryption and decryption speed on ETRU to that of NTRU. Recall in Section 3.2.6 we developed expressions to count the number of multiplications and divisions involved in the encryption and decryption process. We develop similar expressions for ETRU encryption and decryption algorithms.

First recall that for any two Eisenstein integers $\alpha = a + b\omega$ and $\beta = c + d\omega$ we have $\alpha\beta = (ac - bd) + (ad + bc - bd)\omega$. Since bd is calculated twice, multiplication of two Eisenstein integers involves 4 integer multiplications. We count the number of multiplications involved in our division algorithm to be 16, or about 4 times slower than an Eisenstein integer multiplication.

To encrypt a message we compute

$$e \equiv pg * h + m \mod q$$

which involves $N^2 + N$ Eisenstein integer multiplications and N Eisenstein integer reductions. If we take Eisenstein integer reduction to be 4 Eisenstein integer multiplications, there are a total of

$$(N^2 + N) + 4N = N^2 + 5N$$

Eisenstein integer multiplications or

$$4(N^2 + 5N) = 4N^2 + 20N \tag{4.4.3}$$

integer multiplications involved in the ETRU encryption algorithm.

To decrypt a message we compute

$$a \equiv f * e \mod q$$

which involves N^2 Eisenstein integer multiplications and N reductions modulo q. We then compute

$$m \equiv F_p * a \mod p$$

which involves another N^2 Eisenstein integer multiplications and N reductions. Hence there are a total of

$$4 * (2N^2 + 4 * 2N) = 8N^2 + 32N \tag{4.4.4}$$

integers multiplications involved in the ETRU decryption process. Notice that as for NTRU, the encryption and decryption for ETRU are both $O(N^2)$.

We used experimental data to test the speed of Eisenstein integer multiplication to integer multiplication. Our implementation of NTRU and ETRU stores polynomials and Eisenstein integers as objects. The multiplication of two coefficients of an integer polynomial takes much longer than the multiplication of two integers. Therefore we compared the time to multiply the first two coefficients of ten million randomly generated polynomials in $\mathbb{Z}[x]$ with the time to multiply the first two coefficients of ten million randomly generated polynomials in $\mathbb{Z}[\omega][x]$. We found that Eisenstein integer multiplication was approximately 4.75 times slower than integer multiplication which is slower than expected. This is likely due to our implementation as we store both Eisenstein integers and polynomials as objects, further slowing operations in $\mathbb{Z}[\omega][x]$.

We adjust expressions (4.4.3) and (4.4.4) for our implementation. If we take one Eisenstein integer multiplication to be 4.75 integer multiplications the we have

$$4.75(N^2 + 5N) = 4.75N^2 + 23.75N \tag{4.4.5}$$

				Time (s)		Formula		Ratio	
N	d_f	d_g	d_ϕ	Encrypt	Decrypt	Encrypt	Decrypt	Encrypt	Decrypt
41	4	4	4	13.6346	26.173	8958.5	17528	657.04	669.66
47	4	4	4	17.65	34.09	11609	22772	657.74	668.06
53	4	4	4	22.21	43.09	14602	28700	657.57	666.07
83	5	5	5	52.69	104.62	34694	68600	658.44	655.72
97	5	5	5	71.82	140.91	46997	93072	654.36	660.50
100	6	6	6	76.91	149.65	49875	98800	648.46	660.19
113	6	6	6	96.59	191.4	63337	125600	655.70	656.21
133	6	6	6	132.88	263.06	87182	173100	656.07	658.03
150	7	7	7	168.67	334.34	110438	219450	654.76	656.36
							Average:	655.57	661.20

Table 4.1: ETRU Encryption and Decryption Speed with $p = 2 + 3\omega$ and $q = 81\omega = (1-\omega)^8$. We record the time it takes to encrypt and decrypt 10000 messages. We also record the ratio of the number of elementary operations for encryption and decryption given by equations (4.4.5) and (4.4.6) to the time it took for encryption and decryption of the 10000 messages.

integer multiplications for encryption and

$$4.75 * (2N^2 + 4 * 2N) = 9.5N^2 + 38N \tag{4.4.6}$$

integer multiplications for decryption.

We randomly generated 10000 messages and recorded the amount of time it took to encrypt and decrypt the messages in Table 4.1 for various parameter sets. We notice that the decryption process takes about twice as long as the encryption process. We also compare the ratio of the number of operations given by expressions (4.4.5) and (4.4.6) to the time it took to encrypt and decrypt the messages.

The ratios are fairly consistent as can be seen in Figure 4.5. The average ratio for encryption is 655.57 which is very close to 661.20, the average ratio of decryption. These numbers are very close to 661.80, the average ratio for NTRU encryption in Table 3.1. The average ratio for NTRU decryption is slightly lower at 624.89. We conclude that it is valid use expressions (4.4.5) and (4.4.6) to compare the encryption

and decryption speed of ETRU to that of NTRU for our implementation. However expressions (4.4.3) and (4.4.4) are valid for more efficient implementations.

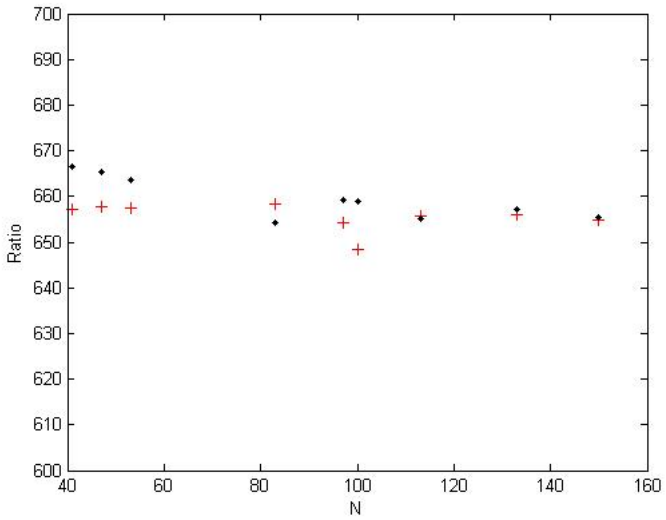

Figure 4.5: Plot of the ratios in Table 4.1. The crosses represent encryption and the dots represent decryption.

Recall that the encryption and decryption of NTRU are both of order $O(N^2)$. Similarly the encryption and decryption of ETRU are also both of order $O(N^2)$. Recall the number of elementary operations for NTRU decryption given by equation (4.4.4) is $2N^2+2N$ therefore ETRU decryption is approximately 4 times slower for the same N. However, in Chapter 5 we will show that we can take the ETRU parameter N to be smaller than the NTRU parameter N for similar levels of security. Consequently the speed of ETRU decryption is comparable to, and sometimes even better, that of NTRU for similar security levels.

Similar to the NTRU case, one could also apply algorithms to speed up multiplications of polynomials as described in [34].

Chapter 5

Security Analysis of ETRU

We now modify the main attacks against NTRU we saw in Section 3.3 for ETRU. We then use a combination of theoretical comparison and experimental data to compare the security of ETRU to that of NTRU.

5.1 Alternate Keys

Recall in Section 3.3.1 we saw that any rotation of the NTRU private key f (or $-f$) can encrypt and decrypt the same messages as f. Following similar arguments, any rotation f' of the ETRU private key f can encrypt and decrypt the same messages as f as well as any rotation of μf where $\mu \in \{\pm 1, \pm\omega, \pm\omega^2\}$ is a unit.

5.2 Brute Force Attacks

As in the case for classical NTRU, an attacker can recover the private key f by trying all possible $f \in \mathcal{L}_f$ and testing to see if $f * h \mod q$ has small entries or by trying all possible $g \in \mathcal{L}_g$ and testing to see if $g * h^{-1} \mod q$ has small entries.

Recall that the polynomials $g \in \mathcal{L}_g$ of length N contain exactly d_g of each element in $\{\pm 1, \pm\omega, \pm\omega^2\}$ and the rest of the coefficients are zero. We count the number of

elements in $\mathcal{L}_g$. First we choose the locations for the 1's; there are $\binom{N}{d_g}$ possibilities. Next we choose the locations for the -1's; there are $\binom{N-d_g}{d_g}$ possibilities. Continuing this pattern we count the total number of elements in $\mathcal{L}_g$ is:

$$\begin{aligned} |\mathcal{L}_g| &= \binom{N}{d_g}\binom{N-d_g}{d_g}\binom{N-2d_g}{d_g}\binom{N-3d_g}{d_g}\binom{N-4d_g}{d_g}\binom{N-5d_g}{d_g} \\ &= \frac{N!}{(d_g!)^6\,(N-6d_g)!}. \end{aligned} \tag{5.2.1}$$

Since any rotation of the NTRU key pair (f,g) can be used to decrypt messages we can look for any rotation of g. As before we can slightly reduce the search time by fixing the constant term to be 1 and search for the other $N-1$ coefficients.

A comparison of the ETRU key space size to the NTRU key space size is discussed in Section 5.5.2.

5.3 Meet-in-the-Middle Attack Against ETRU

Recall in Section 3.3.3 we saw that the meet-in-the-middle attack against NTRU cut the search time on the keyspace by approximately a square root. We will now look at adapting the meet-in-the-middle attack for ETRU. To simplify our presentation we will assume that N is even and that f has d_f of each element of $\{\pm 1, \pm\omega, \pm\omega^2\}$. In this case the polynomial f has $6d_f$ nonzero terms.

Recall the idea behind the meet-in-the-middle is to search for an $f \in \mathcal{L}_f$ of the form $f = f_1||f_2$. For example we may choose f_1 and f_2 as in Section 3.3.3 where f_1 and f_2 are polynomials of length $N/2$ that contain $3d_f$ nonzero terms. We can think of f_1 as a polynomial of length N with it non-zero coefficients in the first $N/2$ positions and similarly f_2 as a polynomial of length N with it non-zero coefficients in the last $N/2$ positions so we have $f = f_1 + f_2$. Since $f * h \equiv g \mod q$ we have

$$f_1 * h \equiv g - f_2 * h \mod q$$

and hence each of the coefficients of $f_1 * h$ and $-f_2 * h$ will differ by a coefficient of g, that is each coefficient will differ by either $0, \pm 1, \pm\omega$ or $\pm\omega^2$ modulo q.

First we enumerate all the possible f_1's and then we put each f_1 into a bin corresponding to the most significant bits of the first k coordinates of $f_1 * h \mod q$. Next we enumerate all the possible f_2's and check to see if $-f_2 * h \mod q$ corresponds to an occupied bin. We also need to check all the bins that correspond to adding either $\pm 1, \pm\omega$ or $\pm\omega^2$ to the coefficients of $-f_2 * h \mod q$.

Recall from equations (3.3.3) and (3.3.4), the time to enumerate the f_1's and write them to the bin corresponding to $f_1 * h \mod q$ is at most

$$\tau_1 = (\#f_1's) \times (\tau_c + \tau_l)$$

where τ_c is the time it takes to compute the convolution $f_1 * h \mod q$ and τ_l is the time it take to look up and write to a bin. Next we need to enumerate the f_2's and check the bins corresponding to adding 0, ± 1, $\pm\omega$ and $\pm\omega^2$ to $-f_2 * h \mod q$. Under the assumption that each bin has the same probability of being occupied by an $f_1 * h \mod q$, the time for this step is at most

$$\tau_2 = (\#f_2's) \times \left(\tau_c + B^A \tau_l + B^A \frac{\#f_1's}{B} \tau_c\right) \tag{5.3.1}$$

where B is the total number of possible bins and A is the probability the leading bit will flip when adding one of ± 1,$\pm\omega$ or $\pm\omega^2$.

We want to determine how big equation 5.3.1 is. To accomplish this we need to determine how to label a bin corresponding to an Eisenstein integer. We also need to determine the probability the leading bit will flip when adding one of ± 1,$\pm\omega$ or $\pm\omega^2$ and we need to count the number of f_1's and f_2's.

It is clear that the number of possible f_1's is equal to the number of possible f_2's. As there are polynomials of the form $f = f_1||f_2$ that are not in $\mathcal{L}_f$ we expect the total number of $f's$ of the form $f = f_1||f_2$ to be larger than the number of elements

in $\mathcal{L}_f$. Recall the size of the keys space $\mathcal{L}_f$ is given by equation (5.2.1). Therefore

$$(\#f_1's) \times (\#f_2's) > |\mathcal{L}_f| = \frac{N!}{(d_g!)^6\,(N-6d_g)!}$$

so a lower bound for the number of f_1's is:

$$(\#f_1's) = (\#f_2's) > \sqrt{\frac{N!}{(d_g!)^6\,(N-6d_g)!}}.$$

Next we need to consider how to convert an Eisenstein integer $\alpha = a + b\omega$ to a binary representation. This is much more difficult than the integer case. One method is to represent α as two binary numbers, the binary representation of a and b. For example suppose $k = 2$ and the first two coefficients of $f_1 * h \mod q$ are $2 + 5\omega + (7 + 1\omega)x$ and suppose that the integers a and b for any Eisenstein integer $\alpha = a + b\omega$ modulo q can be represented by 3 bits, then

$$\begin{aligned} 2 &= (\underline{0}10)_2 \\ 5 &= (\underline{1}01)_2 \\ 7 &= (\underline{1}11)_2 \\ 1 &= (\underline{0}01)_2 \end{aligned}$$

and we store $f_1 * h$ in the bin labeled 0110. Some difficulties include determining the number of bits needed to store a and b for an Eisenstein integer $\alpha \equiv a + b\omega$ modulo q and determining whether the most significant bit of either a or b changes when we add either $\pm 1, \pm\omega$ or $\pm\omega^2$ to $\alpha \mod q$.

It is difficult to determine whether the leading bit of a or b flips when we represent $\alpha \equiv a + b\omega \mod q$ in the hexagonal fundamental domain illustrated in Figure 4.2 because the axes of the hexagon are not parallel to $\vec{1}$ and $\vec{\omega}$. Note however, it is relatively easy to convert from the hexagonal fundamental domain to the parallelogram fundamental domain illustrated in Figure 4.3. It is much easier to determine the number of bits needed to store a and b and visualize and count the cases where the

leading bit flips if we store the Eisenstein integers in the parallelogram fundamental domain.

There are two fundamentally different cases. Let $q = a + b\omega$. The first case is when $d = \gcd(a, b) = 1$, then the parallelogram is a line and the distinct equivalence classes modulo q can be represented by the integers in the set $\{0, 1, \ldots, d(q) - 1\}$ which we can then represent as binary numbers. In this case any $\alpha \mod q$ can be represented by a single binary number of $\log_2 \lceil d(q) \rceil$ bits and there are a total of $B = 2^k$ possible bins. The second case is when $d = \gcd(a, b) \neq 1$. In this case an Eisenstein integer $\alpha \equiv u + v\omega \mod q$ with $0 \leq u < x = \frac{d(q)}{d}$ and $0 \leq v < d$ can be represented by a pair of binary numbers, the binary representations of u and v. Therefore there are a total of $B = 2^{2k}$ possible bins.

First we consider the case where $d = 1$. We need to determine when the leading bit changes if we add $\pm 1, \pm\omega$ or $\pm\omega^2$. As in the trinary case for NTRU, there are only four cases where the leading bit changes if we add ± 1. The difficulty lies in adding $\pm\omega$. Recall that $\omega^2 = -1 - w$ hence if we can add $\pm\omega$ we understand the consequences of adding $\pm\omega^2$.

The proof of Theorem 4.3.3 gives us an algorithm to reduce $\beta = n + m\omega$ modulo $q = a + b\omega$, that is $\beta \equiv u + v\omega \mod q$ for some $0 \leq u < x$ and $0 \leq v < d$ where $d = \gcd(a, b)$ and $x = d(q)/d$. Let $\beta = \omega$, hence $n = 1$ and $m = 0$. We assumed $d = 1$ which implies $x = d(q)$ and it follows that $v = 0$. Recall that equation 4.3.1 gives us a formula to find u. We let $n = v + kd$, hence $k = 1$. Since $d = \gcd(a, b)$ we can find and s and t such that $sb + ta = d$. We have

$$m - ksa - kta + ktb \equiv u \mod x$$

and it follows that

$$u \equiv -sa - ta + tb \mod d(q)$$

and therefore $-sa - ta + tb \mod x \equiv \omega \mod q$. Consequently we can easily calculate $\beta \mod q$.

Example 5.3.1 For example in the case where $q = 91 + \omega$ we have $d(q) = 8191$ and hence the equivalence classes modulo q can be represented by $\{0, \ldots, 8190\}$. Since $2^{13} = 8192$ we represent the equivalence classes with binary numbers of 13 bits. We find that $(1)(91) + (-90)(1) = d$ hence we take $t = 1$ and $s = -90$. It follows that $\omega \equiv 8100 \mod q$ and therefore $-\omega \equiv 91 \mod q$. Next we determine when the leading bit changes if we add $\pm 1, \pm\omega$ or $\pm\omega^2$. When we add 1 the leading bit changes for 4095 and 8190, when we add -1 the leading bit changes for 0 and 4096, when we add ω the leading bit changes for the integers on the intervals $[0, 90]$ and $[4096, 4186]$, when we add ω the leading bit changes for the integers on the intervals $[4005, 4095]$ and $[8100 - 8190]$, when we add $-\omega^2 = \omega + 1$ the leading bit changes on the intervals $[0, 89]$ and $[4096, 4185]$ and finally when we add $\omega^2 = -\omega - 1$ the leading bit changes on the intervals $[4003, 4095]$ and $[8101, 8190]$. Hence leading bit changes on the intervals $[0, 90]$, $[4005, 4095]$, $[4096, 4186]$ and $[8100, 8190]$ for a total of $4 \times 91 = 364$ elements. Therefore the leading bit changes for 364 out of the 8191 equivalence classes modulo q if we add $\pm 1, \pm\omega$ or $\pm\omega^2$, hence $A = \frac{364}{8191} \approx 0.0444$.

In this example we determined $A \approx 0.044$. Recall for the meet-in-the-middle attack against NTRU trinary keys described in Section 3.3.3 there are only 4 possibilities a leading bit will flip when adding ± 1 hence the probability a leading bit will flip is $A = 4/q$. In the case where $q = 2048$ we always have $A \approx 0.0020$ which is much smaller than the A in our example. Note the A in our example is actually one of the best cases; we can find a q such that A is much higher.

Example 5.3.2 Take $q = 11 + 86\omega$ which is an Eisenstein prime with $d = 1$ and $dq = 6571$. We need 13 bits to represent an Eisenstein integer modulo q therefore elements on the interval $[0, 4095]$ have leading bit zero and elements on the interval $[4096, 6570]$ have leading bit one. The leading bit for 4095 and 6570 changes when we add 1 and the leading bit for 0 and 4096 changes when we add -1. We find $\omega \equiv 3591 \mod q$ and $-\omega \equiv 2980 \mod q$. In this case the leading bit on the intervals

$[505, 2979]$ and $[4096, 6570]$ change when we add ω, and the leading bit on the intervals $[1116, 3590]$ and $[4096, 6570]$ change when we add $-\omega$. Note that adding $\pm\omega^2$ does not lead to any additional elements where the leading bit changes. Therefore the leading bit on 0 and the elements on the intervals $[505, 3590]$ and $[4096, 6570]$ changes when adding one of ± 1, $\pm\omega$ or $\pm\omega^2$. This is a total of 5562 elements. Therefore the probability that the leading bit changes when adding one of ± 1, $\pm\omega$ or $\pm\omega^2$ is $A = \frac{5562}{6571} \approx 0.8464$

We will calculate the size of the expression 5.3.1 using the q in Example 5.3.2. In this case we have $B = 2^k$ and $A = 0.8464$. We will choose k as in Section 3.3.3 so that $2^k \approx 100 \times (\#f_1' s)$ hence $\frac{\#f_1' s}{B} = \frac{\#f_1' s}{2^k} \approx 1/100$. We have

$$\tau_2 \approx (\#f_2' s) \times \left(\tau_c + (100 \times (\#f_1' s))^A \tau_l + (100 \times (\#f_1' s))^A \frac{\tau_c}{100} \right),$$

hence the leading term is $(\#f_2' s) \times (100 \times (\#f_1' s))^A$. Since $A = 0.8464$, this is approximately of order $(\#f_2' s) \times (\#f_1' s)$ which is comparable to the size of the keyspace. Therefore, for this choice of q, the meet-in-the-middle attack does not speed up the search on the key space over a brute force attack.

We will now consider a meet-the-middle attack on a parallelogram, which is the case where $d \neq 1$.

Example 5.3.3 Let $q = 81\omega = (1-\omega)^8$ so $d(q) = 81^2 = 6561$. An Eisenstein integer can be written as $\alpha \equiv a + b\omega \mod q$ with $0 \leq a < 81$ and $0 \leq b < 81$ as illustrated by the fundamental domain of q in Figure 5.1. Notice that the four vertices of the parallelogram are elements of $\mathcal{L}(q)$. The leading bit of b changes when we add one of ± 1, $\pm\omega$ or $\pm\omega^2$ if we cross the horizontal lines in the diagram. There are 4 possibilities where this happens: when $b = 0$, $b = 63$, $b = 64$ or $b = 80$. Hence the probability that the leading bit of b changes is $\frac{4}{81}$. By similar arguments the probability that the leading bit of a changes is also $\frac{4}{81}$. Therefore the probability of any leading bit flipping is $A = \frac{4 \times 81}{81^2} = \frac{4}{81} \approx 0.0494$.

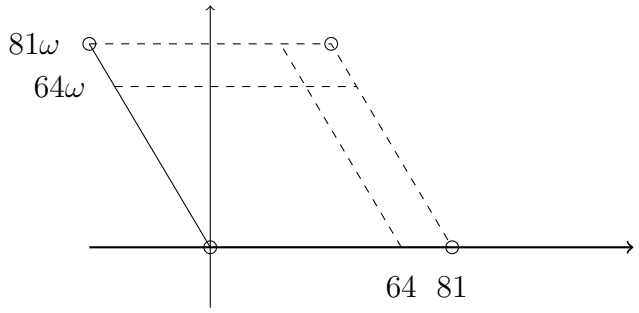

Figure 5.1: Fundamental domain for $q = 81\omega$. The open dots represent the lattice $\mathcal{L}(q)$. A leading bit of an element $\alpha \equiv a + b\omega \mod q$ changes when we add one of ± 1, $\pm\omega$ or $\pm\omega^2$ if we cross one of the edges of the parallelogram or if cross the line with slope ω passing through 64 or the horizontal line passing through 64ω.

Note however, in general the top vertices of the parallelogram are not points in the lattice $\mathcal{L}(q)$ and this can greatly increase the probability of A. In fact Example 5.3.3 is also one of the best cases. For example a q with long narrow parallelogram as a fundamental domain may have a much higher A, similar to the case where $d = 1$.

In general the probability that a leading bit changes depends highly on the choice of q, and for ETRU q can be chosen to make the probability $A > 0.85$. The probability A is much higher for ETRU than for the trinary case of NTRU where the probability that a leading bit changes is always $4/q$. Therefore the meet-in-the-middle attack against ETRU is not as effective as against NTRU. The effectiveness of the meet-in-the-middle attack again ETRU is highly dependent on our choice of q and therefore we can choose a q where this meet-in-the-middle attack does not significantly speed up the search of the key space over a brute force attack.

5.4 Lattice Attacks Against ETRU

In this section we adapt the lattice attack seen in Section 3.3.4 to ETRU.

5.4.1 The ETRU Lattice $\mathcal{L}^{ET}$

Recall that the NTRU lattice $\mathcal{L}^{NT}$ is spanned by the rows of

$$\mathcal{B}^{NT} = \begin{bmatrix} \lambda I & H \\ 0 & qI \end{bmatrix}$$

where H is the circulant matrix corresponding to the NTRU public key h and $\lambda \in \mathbb{R}$ is the balancing constant.

We need to adapt the NTRU lattice for ETRU. For each Eisenstein integer $\alpha = a + b\omega$ we define

$$\langle \alpha \rangle = \begin{bmatrix} a & b \\ -b & a-b \end{bmatrix}.$$

Notice that for any Eisenstein integer $\beta = c + d\omega$ we have

$$\begin{bmatrix} c & d \end{bmatrix} \begin{bmatrix} a & b \\ -b & a-b \end{bmatrix} = \begin{bmatrix} ac - bd & bc + ad - bd \end{bmatrix}$$

which is the vector representation of the multiplication $\beta\alpha$.

For each $n \times n$ matrix A with entries that are Eisenstein integers we will set $\langle A \rangle$ to be the $2n \times 2n$ matrix with each entry a_{ij} replaced with the 2×2 matrix $\langle a_{ij} \rangle$. The *ETRU lattice* $\mathcal{L}^{ET}$ is generated by the rows of the $4N \times 4N$ matrix

$$\mathcal{B}^{ET} = \begin{bmatrix} \lambda \langle I \rangle & \langle H \rangle \\ 0 & \langle qI \rangle \end{bmatrix}.$$

Notice that for every vector $[f' \quad u] \in \mathbb{Z}^{4N}$ with $f', u \in \mathbb{Z}^{2N}$ we have

$$[f' \quad u]\mathcal{B}^{ET} = [\lambda f' \quad f' \langle H \rangle + \langle qI \rangle u] = [\lambda f' \quad g']$$

with $g' \equiv f' * h \mod q$. Therefore the vector $[\lambda f \quad g]$ is contained in the lattice. The vector $[\lambda f \quad g]$ is a relatively short vector in the lattice, therefore we can use basis reduction techniques such as the LLL or BKZ algorithms to try and recover the vector $[\lambda f \quad g]$ and hence recover our ETRU private key f.

Note that the implementations of LLL and BKZ in Shoup's NTL library [32] find short vectors relative to the Euclidean norm. However the vector $[a \quad b]$ corresponding the Eisenstein integer $\alpha = a + b\omega$ has Euclidean norm $\sqrt{a^2 + b^2}$ while α has norm $\sqrt{a^2 + b^2 - ab}$ hence the norm is not preserved when converting an Eisenstein integer to vector from. In particular the units $\pm\omega^2$ have Euclidean norm $\sqrt{2}$ which is greater than their algebraic norm 1. Since we have chosen the polynomial f to have d_f+1 1's, d_f of each of the other five units and the rest on the coefficients 0 and the polynomial g is chosen to have d_g of each unit, the polynomial $\lambda f + g$ has norm

$$\tau_e = \sqrt{\lambda^2(6d_f + 1) + 6d_g}$$

however since the units $\pm\omega^2$ have Euclidean norm $\sqrt{2}$ and the other four units have Euclidean norm 1, the norm of the vector $[\lambda f \quad g]$ is

$$\begin{aligned} \tau_c &= \sqrt{\lambda^2[(4d_f + 1)(1)^2 + 2d_f(\sqrt{2})^2] + (4d_g)(1)^2 + 2d_g(\sqrt{2})^2} \\ &= \sqrt{\lambda^2(8d_f + 1) + 8d_g}. \end{aligned}$$

This is still a relatively short vector in the lattice $\mathcal{L}^{ET}$ and the LLL or BKZ algorithms can still be used to find a vector of length τ_c.

5.4.2 Balancing Constant λ

As before the lattice constant λ is chosen to maximize the efficiency of finding short vectors in the lattice. As in Section 3.3.4 we want to choose λ to maximize the ratio $s/\left|\tau\right|_2$, where s is the length of the shortest expected vector in the lattice and τ is the actual shortest vector.

We let τ be the length of our target vector $[\lambda f \quad g]$, hence

$$\tau = \sqrt{\lambda^2 \left\|f\right\|^2 + \left\|g\right\|^2}.$$

Here $\|f\|$ denotes the Euclidean norm of the vector representation of f. Recall the

Gaussian heuristic (2.3.3) states that the length of the expected shortest vector is

$$s = \sqrt{\frac{d}{2\pi e}}(\det \mathcal{L})^{1/d}$$

where d is the dimension of the lattice. The dimension of $\mathcal{L}^{ET}$ is $4N$ and it determinant is $\lambda^{2N} d(q)^N$ therefore the shortest expected vector is

$$s = \sqrt{\frac{2N \|q\| \lambda}{\pi e}}.$$

Therefore we want to maximize the ratio

$$s/\tau = \sqrt{\frac{2N \|q\| \lambda}{\pi e(\lambda^2 \|f\|^2 + \|g\|^2)}}$$

this is the same as maximizing the ratio

$$\frac{\lambda}{\lambda^2 \|f\|^2 + \|g\|^2}.$$

Notice that this is the same expression as we found in section 3.3.4 hence we choose $\lambda = \|g\| / \|f\|$. Therefore in the case where $\|f\| \approx \|g\|$ we take $\lambda = 1$.

Note that depending on our choice of d_f and d_g, $\tau = \sqrt{\lambda^2 \|f\|^2 + \|g\|^2}$ may not be the length of the actual shortest vector in $\mathcal{L}^{ET}$. By similar arguments to Proposition 3.3.1 the vector $[\underbrace{(\lambda, 0), \ldots, (\lambda, 0)}_{N}, \underbrace{(0, 0), \ldots, (0, 0)}_{N}]$ which has Euclidean norm $\sqrt{N\lambda^2}$ is in the lattice. However, as in the case with NTRU, this does not seem to affect our success in finding a target vector.

5.4.3 A Norm-Preserving ETRU Lattice $\mathcal{L}^{ET^*}$

Recall basis reduction algorithms such as LLL and BKZ are more successful at finding a shortest vector in a lattice if the length of a shortest vector is much shorter than the shortest expected vector in the lattice. We saw in Section 5.4.1 that the norm the polynomial $\lambda f + g$ is shorter than our target vector $[\lambda f \quad g]$. We will now consider

the ETRU lattice presented in [29] where the vector norms are preserved, that is our target vector will have the same norm as the polynomial $\lambda f + g$.

Every Eisenstein integer $\alpha = a + b\omega$ can be written as a complex number $\alpha = (a - \frac{1}{2}b) + (\frac{\sqrt{3}}{2}b)i = c + di$ and can be represented as the vector $\alpha_c = [c \quad d]$. To convert an Eisenstein integer $\alpha = a + b\omega$ into complex form we must apply the change of basis matrix is

$$z = \begin{bmatrix} 1 & 0 \\ -1/2 & \sqrt{3}/2 \end{bmatrix}.$$

We can see that $[a \quad b]Z = [a - (1/2)b \quad (\sqrt{3}/2)b] = \alpha_c$ which is the vector representation of the complex form of α.

For each complex number $\alpha = a + bi$ we will define

$$(\alpha) = \begin{bmatrix} a & b \\ -b & a \end{bmatrix}.$$

Notice that if $\beta = c + di$ then $[c \quad d](\alpha)$ is the vector representation of the multiplication $\beta\alpha$.

As before, for each $n \times n$ matrix A with entries that are complex numbers we will set (A) to be the $2n \times 2n$ matrix with each entry a_{ij} replaced with the 2×2 matrix (a_{ij}). The ETRU lattice $\mathcal{L}^{ET^*}$ is generated by the following $4N \times 4N$ matrix

$$\mathcal{B}^{ET^*} = Z \begin{bmatrix} \lambda I & (H) \\ 0 & (qI) \end{bmatrix}.$$

where Z is the $4N \times 4N$ change of basis matrix

$$Z = \begin{bmatrix} z & 0 & \dots & 0 \\ 0 & z & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & z \end{bmatrix}.$$

Let the vectors $f', u \in \mathbb{Z}^{2N}$ be vector representations of two polynomials of length N with Eisenstein integer coefficients. The multiplication $[f' \quad u]Z = [f'_c \quad u_c]$

converts the vector $[f' \quad u] \in \mathbb{Z}^{4N}$ to complex form. Therefore,

$$[f' \quad u]\mathcal{L}^{ET^*} = [f'_c \quad u_c] \begin{bmatrix} \lambda I & (H) \\ 0 & (qI) \end{bmatrix} = [\lambda f'_c \quad f'_c\,(H) + (qI)\,u_c] = [\lambda f'_c \quad g'_c]$$

with f'_c and g'_c the complex form of the polynomials f' and g' such that $g' \equiv f' * h \mod q$. Hence the the vector $[\lambda f_c \quad g_c]$ is in the lattice and has the same norm as the polynomial $\lambda f + g$. Since $[\lambda f_c \quad g_c]$ is a relatively short vector in the lattice we can use lattice reduction techniques to try and recover our private key.

By similar arguments to 5.4.2 we want to take $\lambda = \|g_c\| \,/\, \|f_c\|$.

5.4.4 Comparison Between $\mathcal{L}^{ET}$ and $\mathcal{L}^{ET^*}$

Recall in Section 5.4.1 we saw the norm of our target vector $[\lambda f_c \quad g_c]$ in the lattice $\mathcal{L}^{ET*}$ is

$$\tau_e = \sqrt{\lambda^2(6d_f + 1) + 6d_g}.$$

and the norm of $[\lambda f \quad g]$ in $\mathcal{L}^{ET}$ is

$$\tau_c = \sqrt{\lambda^2(8d_f + 1) + 8d_g}.$$

Notice that the target vector for $\mathcal{L}^{ET}$ is slightly longer than our target vector for $\mathcal{L}^{ET^*}$ therefore we expect basis reduction techniques to be more successful on $\mathcal{L}^{ET^*}$, however $\mathcal{L}^{ET^*}$ is a non integral lattice so we expect the LLL and BKZ algorithms to be slower. We gather experimental data to compare the security of the two ETRU lattices.

In Table 5.1 we record the results of our experiments on $\mathcal{L}^{ET}$. We ran experiments for randomly generated ETRU keys for different sets parameters. For each parameter set we hold p and q constant taking $p = 2 + \omega$ and $q = 81\omega = (1 - \omega)^8$. We choose $d_f = d_g$ so we have $\|f\| \approx \|g\|$ and therefore we take $\lambda = 1$. We run the LLL and BKZ algorithms on the ETRU lattices $\mathcal{L}^{ET}$ and record whether we found a vector the same length as our target vector τ_c. In this case we usually have found a rotation

			LLL			BKZ		
N	d_f	d_g	Target %	Trials	Avg Time (s)	Target %	Trials	Avg Time (s))
17	2	2	99.9	1000	0.53	100	1000	1.05
19	2	2	100	1000	0.77	100	1000	1.62
21	3	3	94.5	1000	1.20	99.5	1000	3.27
23	3	3	47.2	1000	1.46	100	1000	3.46
25	3	3	8.4	1000	1.91	98.0	1000	4.82
29	3	3	0	1000	2.71	100	1000	10.11
31	3	3	0	1000	3.25	99.9	1000	16.44
33	3	3	0.1	1000	3.85	99.4	500	21.69
35	4	4	0	1000	4.43	98.8	500	39.45
41	4	4	0	1000	6.70	1.6	500	80.91
47	4	4				0	100	139.45

Table 5.1: Lattice reduction on $\mathcal{L}^{ET}$ with $p = 2+3\omega$ and $q = 81\omega = (1-\omega)^8$ using Shoup's [32] LLL_FP and BKZ_FP functions for $\delta = 0.99$ and blocksize $\beta = 10$.

of μf, where μ is a unit, which can be used to decrypt the same messages as f. We also record the average time per trial. We used the LLL_FP and BKZ_FP functions in Victor Shoup's C++ NTL library available at [32] choosing $\delta = 0.99$. The block size for the BKZ algorithm is chosen as the default size of $\beta = 10$.

Next we want to apply LLL and BKZ to $\mathcal{L}^{ET^*}$ for the same parameter sets. However, $\mathcal{L}^{ET^*}$ is not an integral lattice and we can not directly use the LLL and BKZ algorithms in Shoup's NTL library [32]. We take the approach given in [1] and [2] for reducing lattices by means of approximation. The idea is to let

$$B = k\mathcal{B}^{ET^*}$$

for some integer constant k then round each entry in B to the nearest integer to get an integer lattice. We can now run the LLL or BKZ algorithm on B to get a LLL-reduced basis B'. Next we apply U to $\mathcal{B}^{ET^*}$ where U is the unimodular transformation such that $B = UB'$. Hence $U\mathcal{B}^{ET^*}$ is an approximation to the LLL reduced basis for $\mathcal{L}^{ET^*}$. Note that the functions LLL_FP and BKZ_FP in Shoup's NTL library both return

			LLL			BKZ		
N	d_f	d_g	Target %	Trials	Avg Time (s)	Target %	Trials	Avg Time (s)
15	2	2	100	500	4.81	100	200	16.32
19	2	2	100	500	7.76	100	200	20.39
21	3	3	96	500	12.07	99.5	200	27.06
23	3	3	55.2	500	16.48	100	200	36.43
25	3	3	7.8	500	21.88	99.5	200	47.16
29	3	3	0.4	500	38.18	100	200	134.28
31	3	3	0	500	48.04	100	200	149.07
33	3	3				100	100	211.205
35	4	4				97.0	100	331.299
41	4	4				8	50	635.686
47	4	4				0	50	1074.98

Table 5.2: Lattice reduction on $\mathcal{L}^{ET^*}$ with $p = 2 + 3\omega$ and $q = 81\omega = (1-\omega)^8$ using Shoup's [32] LLL_FP and BKZ_FP functions with $\delta = 0.99$ and blocksize $\beta = 10$.

the unimodular matrix U.

An algorithm for choosing k is described in [2], however this gives a very large and impractical k. For example in the case where $N = 17$, $p = 2 + 3\omega$, $q = 81\omega$ and $d_f = d_f = 2$ it gives k to be of order 10^{96}. instead we chose $k = 1000000$ as we did not notice an improved speed for choosing a smaller k, nor did we see improved results for taking a larger k.

It is clear that reduction on $\mathcal{L}^{ET^*}$ will be slower than reduction on $\mathcal{L}^{ET}$ as not only we are working with much larger integers, there are more steps involved including converting Eisenstein integers to integer coordinates and back and multiplication of $4N \times 4N$ matrices such as when we generate $\mathcal{B}^{ET^*}$ and calculate $U\mathcal{B}^{ET^*}$.

In Table 5.2 we record the results of our experiments on $\mathcal{L}^{ET^*}$ for the same parameter sets. We record whether we found a vector the same length as our target vector τ_e as well as the average time per trial. As we can see from Tables 5.1 and 5.2 the BKZ algorithm is slower however it gives better results than the LLL algorithm.

The lattices $\mathcal{L}^{ET}$ and $\mathcal{L}^{ET^*}$ seem to have similar levels of lattice security however the lattice attacks on $\mathcal{L}^{ET^*}$ are much slower. For example at $N = 21$ LLL finds a target vector 94.4% of the time on $\mathcal{L}^{ET}$ and 96% of the time on $\mathcal{L}^{ET^*}$. The success of the LLL algorithm drops sharply on both $\mathcal{L}^{ET}$ and $\mathcal{L}^{ET^*}$ for $N > 21$. For $N > 29$ the LLL algorithm rarely finds a target vector on both $\mathcal{L}^{ET}$ and $\mathcal{L}^{ET^*}$. Similarly the BKZ algorithm is very successful for $N \leq 35$ the its success drops sharply and rarely finds a target vector for $N > 47$ on both $\mathcal{L}^{ET}$ and $\mathcal{L}^{ET^*}$.

Therefore $\mathcal{L}^{ET}$ is the better choice for the attacker as lattice reduction is much faster and $\mathcal{L}^{ET}$ has a similar level of security as $\mathcal{L}^{ET^*}$.

5.5 Comparing NTRU and ETRU

We now compare NTRU and ETRU on the common criteria we have developed in Chapters 4 and 5.

5.5.1 Lattice Security

We first compare the lattice security of NTRU to that of ETRU. We ran experiments for randomly generated NTRU keys for given parameter sets. For each parameter set we take $p = 3$ and $q = 2048$. We choose $d_f = d_g$ so we have $\|f\| \approx \|g\|$. We ran Shoup's [32] LLL_FP and BKZ_FP algorithms with $\delta = 0.99$ and blocksize $\beta = 10$ and recorded when we found a vector of length of our target vector $\tau = \sqrt{2d_f + 1 + 2d_g}$. In this case we have usually found a rotation of f which can be used to encrypt and decrypt the same messages as f. We also record the average time per trial it takes to reduce $\mathcal{L}^{NT}$ and check to see if we found a vector of length τ.

Table 5.3 records the result from our experiment. In general the LLL and BKZ algorithms are very successful up to a certain N when their success drops fairly quickly to zero. There are a couple exceptions such as when $N = 57$ and $N = 63$ where both

			LLL			BKZ		
N	d_f	d_g	Target %	Trials	Avg Time (s)	Target %	Trials	Avg Time (s)
43	14	14	100	500	2.29	100	500	3.11
53	17	17	100	500	4.85	100	500	5.44
57	19	19	75.8	500	7.36	72.8	500	11.40
61	20	20	99.6	500	8.33	100	500	9.69
63	21	21	52	500	10.91	52	500	23.57
64	21	21	98.4	500	9.59	100	500	10.72
67	22	22	96.2	500	11.48	100	500	13.72
73	24	24	83.6	500	17.24	98.4	500	19.55
83	27	27	25.4	500	27.86	100	500	38.33
91*	30	30	1.5	200	106.12	76	200	115.50
107*	35	35	0	200	116.72	100	200	209.467
113*	37	37	0	200	110.22	100	50	372.86
127*	42	42				32	50	766.04
131*	43	43				22	50	637.05
137*	45	45				4	50	898.54
142*	47	47				0	50	956.87

Table 5.3: Lattice reduction on $\mathcal{L}^{NT}$ with $p = 3$ and $q = 2048 = 2^{11}$ using Shoup's [32] LLL_FP and BKZ_FP functions with $\delta = 0.99$ and blocksize $\beta = 10$. (* We use the slower G_LLL_FP and G_BKZ_FP functions due to floating point errors)

the LLL and BKZ algorithms seems to have much lower than expected success.

We are able to compare the results with the ETRU lattice $\mathcal{L}^{ET}$ with $q = 81\omega$ from Table 5.1. The lattice security of $\mathcal{L}^{ET}$ for ETRU with polynomials of length N appears to be similar to that of $\mathcal{L}^{NT}$ with $q = 2048$ for NTRU with polynomials of length of approximately $3N$. For example, notice for $\mathcal{L}^{ET}$ the success for the LLL algorithm drops sharply for $N > 21$. Similarly the success for the LLL algorithm on $\mathcal{L}^{NT}$ drops for $N > 67$. For $N \geq 29$ the LLL algorithm is rarely successful in finding the target on $\mathcal{L}^{ET}$ and similarly the LLL algorithm is rarely successful in finding the target on $\mathcal{L}^{NT}$ for $N \geq 91$. The success of the BKZ algorithm drops for $N > 35$ on $\mathcal{L}^{ET}$ and for $N > 113$ on $\mathcal{L}^{NT}$.

			$q = 81\omega$		$q = 11 + 86\omega$	
			LLL	BKZ	LLL	BKZ
N	d_f	d_g	Target %	Target %	Target %	Target %
19	2	2	100	100	99.5	
21	3	3	94.5	99.5	100	
23	3	3	47.2	100	100	
25	3	3	8.4	98.0	98.5	
29	3	3	0	100	27	100
31	3	3	0	99.9	0.5	100
33	3	3	0.1	99.4	0	100
35	4	4	0	98.8		100
41	4	4	0	1.6		75
47	4	4		0		0

Table 5.4: Comparison of the lattice strength of $\mathcal{L}^{ET}$ with $\lambda = 1$ for $q = 81\omega$ and $q = 11 + 86\omega$ with $p = 2 + 3\omega$ using Shoup's [32] LLL_FP and BKZ_FP functions with $\delta = 0.99$ and blocksize $\beta = 10$.

Recall, however, we saw in Section 5.3 that $q = 81\omega$ may be vulnerable a meet-in-the-middle attack. Consequently we also compare the lattice security of $\mathcal{L}^{NT}$ with $q = 2048$ to the lattice security of $\mathcal{L}^{ET}$ with $q = 11 + 86\omega$ which was shown to not be vulnerable to the meet-in-the-middle attack presented in Section 5.3. Note that $d(81\omega) = 6561$ and $d(11 + 86\omega) = 6571$ so their norms are very close. We run experiments, similar to the ones described in Section 5.4.4, on $\mathcal{L}^{ET}$ with $q = 11+86\omega$ and compare the results with the results for our experiments on $\mathcal{L}^{ET}$ with $q = 81\omega$ in Table 5.4.

Notice that the lattice security of $\mathcal{L}^{ET}$ with $q = 81\omega$ is slightly higher than for $q = 11+86\omega$. We conclude our choice of q affects the lattice security of ETRU. Note, however, the lattice security of ETRU with $q = 11 + 86\omega$ and polynomials of length N is still comparable to the lattice security of NTRU with $q = 2048$ and polynomials of length approximately $3N$.

We conjecture that the lattice security of ETRU with polynomials of length N is comparable to the lattice security of NTRU with polynomials of length $3N$. Note

if we take the NTRU parameter N to be twice that of the ETRU parameter N then $\mathcal{L}^{ET}$ and $\mathcal{L}^{NT}$ have the same dimension. We conclude from Tables 5.1 and 5.3 that $\mathcal{L}^{ET}$ has higher lattice security than $\mathcal{L}^{NT}$ of the same dimension.

5.5.2 Parameter Sets

Table 5.5 gives the combinatorial security and storage needed for various NTRU parameter sets with $p = 3$ and $q = 2048$. To maximize combinatorial security for the key space given by equation (3.3.5) (and also to maximize the message space) we take $d = d_f = d_g = d_\phi = \lfloor N/3 \rfloor$. Using equation (3.2.1) we find that the probability of decryption failure is always less than 10^{-4}.

The *plaintext* (or messages) consists of polynomials of length N reduced modulo p. An integer reduced modulo p can be converted to binary and stored in $\lceil \log_2 p \rceil$ bits. Therefore we need approximately $N \lceil \log_2 p \rceil$ bits to store polynomials reduced modulo p. Similarly the *ciphertext* (or encrypted messages) consists of polynomials of length N reduced modulo q. We need approximately $N \lceil \log_2 q \rceil$ bits to store polynomials reduced modulo q. We use equations (3.2.2) and (3.2.3) for an estimate of the encryption and decryption speed of NTRU.

Table 5.6 gives the combinatorial security and storage needed for various ETRU parameter sets with $p = 2 + 3\omega$ and $q = 11 + 86\omega$. We want to choose parameter sets to maximize the size of the key space given by equation (5.2.1). We also want to maximize the message space hence we take $d = d_f = d_g = d_\phi$. Recall that as d_f, d_g and d_ϕ increase the probability of decryption failure also increases. Therefore we take the largest d that gives a probability of decryption failure, as given by Theorem 4.4.1, of less than 10^{-4}.

Recall there are $d(p)$ distinct equivalence classes modulo p. Hence to reduce a Eisenstein integer modulo p we need approximately $\lceil \log_2 d(p) \rceil$ bits. Therefore to store the *plaintext* (or message) we need approximately $N \lceil \log_2 d(p) \rceil$ bits. Similarly we

N	d	Com. Security	P.text(bits)	C.text(bits)	Encryption	Decryption
43	14	5.24×10^{9}	86	473	1935	3784
53	17	1.11×10^{12}	106	583	2915	5724
57	19	1.05×10^{13}	114	627	3363	6612
61	20	8.88×10^{13}	122	671	3843	7564
63	21	2.71×10^{14}	126	693	4095	8064
67	22	2.30×10^{15}	134	737	4623	9112
73	24	5.98×10^{16}	146	803	5475	10804
83	27	1.32×10^{19}	166	913	7055	13944
91	30	1.06×10^{21}	182	1001	8463	16744
107	35	6.31×10^{24}	214	1177	11663	23112
142	47	7.46×10^{32}	284	1562	20306	40470
300	100	2.55×10^{70}	600	3300	90300	180300
400	133	2.55×10^{70}	600	3300	160400	320400

Table 5.5: Combinatorial security, storage of plaintext and ciphertext and speed of encryption and decryption for NTRU parameter sets with $p = 3$ and $q = 2048$ with the probability of decryption failure less than 10^{-4}.

need approximately $N \lceil \log_2 d(q) \rceil$ bits to store the *ciphertext* (or encrypted message). We use equations (4.4.3) and (4.4.4) for an estimate of the encryption and decryption speed of NTRU.

Since the meet-in-the-middle attack against ETRU presented in Section 5.3 does not speed up the search on the key space for $q = 11 + 86\omega$, we will compare the size of the key space of ETRU, given by equation (5.2.1), to the combinatorial security of NTRU, given by equation (3.3.5). Recall in Section 5.5.1 we conjectured that the lattice security of ETRU with polynomials of length N is comparable to the lattice security of NTRU with polynomials of length $3N$. Therefore we begin by comparing NTRU and ETRU parameter sets in Tables 5.5 and 5.6 with the NTRU N approximately 3 times the ETRU N.

For example take ETRU $N = 47$ and NTRU $N = 142$. In Section 5.5.1 we saw that $\mathcal{L}^{NT}$ and $\mathcal{L}^{ET}$ have comparable lattice security. In Tables 5.5 and 5.6 we see that ETRU is more efficient than NTRU in terms of key security, storage and speed.

N	d	Key Space	P.text(bits)	C.text(bits)	Encryption	Decryption
17	2	4.6×10^{10}	51	221	1496	2856
19	2	3.8×10^{11}	57	247	1824	3496
21	3	1.8×10^{14}	63	273	2184	4200
23	3	4.6×10^{15}	69	299	2576	4968
29	3	4.7×10^{18}	87	377	3944	7656
31	3	2.8×10^{19}	93	403	4464	8680
33	3	1.4×10^{20}	99	429	5016	9768
35	4	1.4×10^{24}	105	455	5600	10920
41	4	4.9×10^{26}	123	533	7544	14760
47	4	5.2×10^{28}	141	611	9776	19176
100	6	5.3×10^{51}	300	1300	42000	83200
150	7	2.6×10^{66}	450	1950	93000	184800

Table 5.6: Size of key space, storage of plaintext and ciphertext and encryption and decryption speed for ETRU Parameters sets with $p = 2 + 3\omega$ and $q = 11 + 86\omega$ with probability of decryption failure less than 10^{-4}.

Now let us take ETRU $N = 100$ and NTRU $N = 300$. Since the NTRU N is three times the ETRU N we conclude $\mathcal{L}^{NT}$ and $\mathcal{L}^{ET}$ have comparable lattice security. Tables 5.5 and 5.6 show that ETRU is still more efficient than NTRU in terms of speed and security. However, the key space size of ETRU is approximately 5.3×10^{51} which is significantly smaller than the combinatorial security of NTRU which is approximately 2.55×10^{70}. We can increase the size of the ETRU N which will increase the combinatorial and lattice security of ETRU at the expense of storage and speed.

Let us now compare ETRU $N = 150$ with NTRU $N = 300$. In this case we conjecture the lattice security of ETRU is much stronger than that of NTRU and Tables 5.5 and 5.6 show that the storage needed for ETRU is still less than that of NTRU. However, the key security of ETRU is still less than the combinatorial security of NTRU and the encryption and decryption speed of ETRU is slightly slower than that of NTRU.

In general the storage and speed of ETRU are an improvement over NTRU with

similar lattice security, on the other hand NTRU has stronger key security. However, current NTRU parameter sets, as seen in [21] or [15], take combinatorial security much larger than needed for a given security parameter. Lattice attacks are the more viable attack for an attacker.

Also note that there are other factors such as speed are considered when choosing parameter sets. There is an algorithm described in [13] that speeds up multiplication of polynomials with low Hamming weights. Consequently taking NTRU d_f and d_ϕ to be smaller than $\lfloor N/3 \rfloor$ can speed up the encryption and decryption process.

Chapter 6

Conclusions

In this thesis we have expanded on the work of Nevins, KarimianPour and Miri in [29] in extending NTRU over the Eisenstein integers.

Our contributions include modifying the division algorithm for the Eisenstein integers given in [29] so that it outputs unique representatives of the congruence classes and then implementing ETRU using C++. We developed a model for the probability of decryption failure of ETRU and discussed parameter selection. We modified the meet-in-the-middle and lattice attacks for ETRU. We then implemented lattice attacks on ETRU and NTRU to compare their lattice security. We also used experimental data to compare the encryption and decryption speeds on ETRU to NTRU.

We have seen that ETRU has stronger lattice security than NTRU, requires less storage and is comparable in terms of speed. ETRU, on the other hand, has weaker key security. However, this may not be a major issue as parameter sets for NTRU are chosen to have much larger key security than needed for a given security parameter and lattice attacks are the more viable attack.

There have been various modifications to speed up the encryption and decryption of NTRU, which have not been taken into account here. For example taking q to be a

power of 2 allows for an algorithms for more efficient reduction modulo q [8]. There are also algorithms for high speed multiplication in a truncated polynomial ring [34] and algorithm for speeding up multiplication of polynomials with low Hamming weight [13]. One could apply similar ideas to speed up ETRU.

There are other attacks against NTRU that have not been discussed in this thesis such as a hybrid meet-in-the-middle lattice attack [20] and ciphertext attacks [23]. Padding schemes have been developed for NTRU to protect again ciphertext attacks [19]. Future work could also include modifying these attacks and developing padding schemes for ETRU.

Other work could include developing an efficient algorithm to convert messages (binary) to polynomials modulo p. Our suggestion of taking $p = 2 + 3\omega$ allows seven possibilities for each coefficient. We could also take p to be the prime $1 - \omega$ which has 3 distinct equivalence classes allowing for trinary polynomials.

In conclusion, ETRU improves upon NTRU in terms of lattice security and storage and is comparable to NTRU in terms of speed and may be a good alternative to NTRU.

Appendix A

Sample Code

We provide the code for the division algorithm for the Eisenstein integers described in Theorem 4.1.1. The function takes two Eisentsein integers x and y and outputs $r \equiv x \mod y$.

Recall we can write the Eisenstein integers x and y as complex numbers of the form $x = a + bi$ and $y = c + di$. We can represent the Eisenstein integer x as a vector $[a \quad b]^T$. To follow the algorithm we need to find the inverse of the the Eisenstein integer y. We set the matrix

$$B = \begin{bmatrix} c & -d \\ d & c \end{bmatrix}.$$

The matrix multiplication $B[a \quad b]^T$ is the vector representation of the multiplication yx. Therefore the matrix multiplication $B^{-1}[a \quad b]^T$ is the vector representation of the multiplication $y^{-1}x$.

The following code is written in C++.

```
EInt EInt::mod(EInt x, EInt y){             //x = qy +r  (x \equiv r \mod y)

    EInt r;                                 //return remainder r
    double xc[2],yc[2];                     //Complex form of x and y
    double B[2][2];                         //Matrix  corresponding to b
    double invB[2][2];                      //Inverse of matrix B
    double det;                             //determinant of B
    double v[2][2];
    double p[2][2];                         //stores \rho_1 and \rho_2
```

```
    double q[2][2];                         //stores quotient and remainder
    double n1, n2, temp;

    xc[0] = x.geta() - 0.5*x.getb();        //Convert x to complex coordinates
    xc[1] = x.getb()*sqrt(3.0)*(0.5);
    yc[0] = y.geta() - 0.5*y.getb();        //Convert y to complex coordinates
    yc[1] = y.getb()*sqrt(3.0)*(0.5);

    B[0][0] = yc[0];                        //initialize matrix B
    B[0][1] = (-1)*yc[1];
    B[1][0] = yc[1];
    B[1][1] = yc[0];

    det = B[0][0]*B[1][1] - B[1][0]*B[0][1];  //Find inverse of B
    invB[0][0] = B[1][1]/det;
    invB[0][1] = (-1)*B[0][1]/det;
    invB[1][0] = (-1)*B[1][0]/det;
    invB[1][1] = B[0][0]/det;

    v[0][0] = invB[0][0]*xc[0] + invB[0][1]*xc[1];    //v[0]=[a_1 b_1]
    v[0][1] = invB[1][0]*xc[0] + invB[1][1]*xc[1];
    v[1][0] = v[0][0] + 0.5;                          //v[1]=[a_2 b_2]
    v[1][1] = v[0][1] - 0.5*sqrt(3.0);

    for(int i=0; i<2; i++){
        p[i][0] = v[i][0] - nInt(v[i][0]);
        p[i][1] = v[i][1] - sqrt(3.0)*(nInt(v[i][1]/sqrt(3.0)));
    }

    n1 = p[0][0]*p[0][0] + p[0][1]*p[0][1];           //norm^2 of p0
    n2 = p[1][0]*p[1][0] + p[1][1]*p[1][1];           //norm^2 of p1

    if(nInt(1000000*n1) < nInt(1000000*n2)){          //if d(p1) < d(p2)
        q[0][0] = B[0][0]*p[0][0] + B[0][1]*p[0][1];
        q[0][1] = B[1][0]*p[0][0] + B[1][1]*p[0][1];

        q[1][0] = nInt(v[0][0]);
        q[1][1] = sqrt(3.0)*nInt(v[0][1]/sqrt(3.0));
    }
else if(nInt(1000000*n1) == nInt(1000000*n2)){ //if d(p1) = d(p2)
        v[0][0] = nInt(v[0][0]);
        v[0][1] = sqrt(3.0)*nInt(v[0][1]/sqrt(3.0));

        v[1][0] = nInt(v[1][0])-0.5;
        v[1][1] = sqrt(3.0)*nInt(v[1][1]/sqrt(3.0))+sqrt(3.0)/2;

        if(v[0][0]>v[1][0]){                          //choose right
            q[0][0] = B[0][0]*p[0][0] + B[0][1]*p[0][1];
            q[0][1] = B[1][0]*p[0][0] + B[1][1]*p[0][1];
            q[1][0] = v[0][0];
            q[1][1] = v[0][1];
        }
        else{
            q[0][0] = B[0][0]*p[1][0] + B[0][1]*p[1][1];
            q[0][1] = B[1][0]*p[1][0] + B[1][1]*p[1][1];
            q[1][0] = v[1][0];
```

```
            q[1][1] = v[1][1];
        }
    }
    else{                                               //d(p1)>d(p2)
        q[0][0] = B[0][0]*p[1][0] + B[0][1]*p[1][1];
        q[0][1] = B[1][0]*p[1][0] + B[1][1]*p[1][1];

        q[1][0] = nInt(v[1][0])-0.5;
        q[1][1] = sqrt(3.0)*nInt(v[1][1]/sqrt(3.0))+sqrt(3.0)/2;
    }

    r.init(nInt((q[0][0] + q[0][1]/sqrt(3.0))),nInt(q[0][1]*2/sqrt(3.0)));

    return r;
}
```

We developed the class EInt as the class of Eisenstein intgers. We store an Eisenstein integer $x = a + b\omega$ as an array of two integers. The subroutines $x.geta()$ and $x.getb()$ return the values of a and b repectively. The subroutine of $x.init(s, t)$ sets $x = s + t\omega$.

We also wrote the subroutine $nInt(a)$ to return the nearest integer to a. If $|a - \lfloor a \rfloor| < 0.5$ then the function returns $\lfloor a \rfloor$, otherwise the function returns $\lceil a \rceil$.

Bibliography

[1] J. Buchmann, M. Pohst, *Computing a lattice basis from a system of generating vectors*, EUROCAL '87, Lecture Notes in Computer Science 378, pages 54-63, Springer, 1989.

[2] J. Buchmann, *Reducing lattice bases by means of approximations*, Algorithmic Number Theory, Lecture Notes in Computer Science 877, pages 160-168, Springer, 1994.

[3] M. Coglianese and B.-M. Goi, *MaTRU: A New NTRU-Based Cryptosystem*, INDOCRYPT 2005, Lecture Notes in Computer Science 3797, pages 232-243, Springer-Verlag, 2005.

[4] D. Coppersmith and A. Shamir, *Lattice Attacks on NTRU*, Advances in Cryptology, EUROCRYPT '97, Lecture Notes in Computer Science 1233, pages 52-61, Springer-Verlag, 1997.

[5] D. Dummit and R. Foote, *Abstract Algebra*, Third Edition, John Wiley and Sons Inc., 2004.

[6] M. Euchner and C. P. Schnorr, *Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems*, Fundamentals of Computation Theory, Lecture Notes in Computer Science 529, pages 68-85, Springer-Verlag, 1991.

[7] C. Gentry, *Key Recovery and Message Attacks on NTRU-Composite*, Eurocrypt 2001, Lecture Notes in Computer Science 2045, pages 182-194, Springer-Verlag, 2001.

[8] P. Hirschhorn, J. Hoffstein, N. Howgrave-Graham and W. Whyte, *Choosing NTRUEncrypt Parameters in Light of Combined Lattice Reduction and MITM Approaches*, Applied Cryptography and Network Security, Lecture Notes in Computer Science 5536, pages 437-455, Springer-Verlag, 2009.

[9] J. Hoffstein, J. Pipher and J. H. Silverman, *NTRU: A Ring-Based Public Key Cryptosystem*, Algorithmic Number Theory, Lecture Notes in Computer Science 1423, pages 267-288, Springer-Verlag, 1998.

[10] J. Hoffstein, J. Pipher and J. H. Silverman, *An Introduction to Mathematical Cryptography*, Undergraduate Texts in Mathematics, Springer, 2008.

[11] J. Hoffstein, *Invertibility in Truncated Polynomial Rings*, NTRU Cryptosystems Technical Report 9, Version 1, 1998. Available from: http://www.ntru.com. Accessed Dec. 2010.

[12] J. Hoffstein and J. H. Silverman, *Optimizations for NTRU*, in Public-Key Cryptography and Computational Number Theory, DeGruyter, Berlin, 2000. Available from: http://www.ntru.com. Accessed: Dec. 2010.

[13] J. Hoffstein and J. H. Silverman, *Random Small Hamming Weight Products with Applications to Cryptography*, Discrete Applied Mathematics, Volume 130, Issue 1, Elsevier Science Publishers B. V., 2003. Available from: http://www.ntru.com. Accessed: Dec. 2010.

[14] J. Hoffstein, J. H. Silverman and W. Whyte, *Estimated Breaking Times for NTRU Lattices*, NTRU Cryptosystems Technical Report 12, Version 2, updated 2006. Available from: http://www.ntru.com. Accessed: Dec. 2010.

[15] J. Hoffstein, N. Howgrave-Graham, J. Pipher and W. Whyte, *Practical lattice-based cryptography: NTRUEncrypt and NTRUSign*, The LLL Algorithm: Survey and Applications, pages 349-390, Information Security and Cryptography, Springer-Verlag, 2010.

[16] R. Hogg and E. Tannis, *Probability and Statistical Inference*, Sixth Edition. Prentice-Hall, 2001.

[17] N. Howgrave-Graham, J. H. Silverman and W. Whyte, *Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3*, CT-RSA 2005, Lecture Notes in Computer Science, Volume 3376/2005, pages 118-135, Springer Berlin, 2005.

[18] N. Howgrave-Graham, J. H. Silverman and W. Whyte, *A Meet-in-the-Middle Attack on an NTRU Private Key*, NTRU Cryptosystems Technical Report 4, Version 2, updated 2006. Available from: http://www.ntru.com. Accessed: Dec. 2010.

[19] N. Howgrave-Graham, J. H. Silverman, A. Singer, and W. Whyte, *NAEP: Provable Security in the Presence of Decryption Failures*, Available from: http://www.ntru.com. Accessed: Dec. 2010.

[20] N. Howgrave-Graham, *A Hybrid Lattice-Reduction and Meet-in-the-Middle Attack Against NTRU*, CRYPTO 2007, Lecture Notes in Computer Science 4622, pages 150-169, 2007.

[21] IEEE Computer Society, *IEEE Standard Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices*, IEEE Std 1363.1-2008, The Institute of Electrical and Electronics Engineers, 2009.

[22] K. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory*, Second Edition. New York: Springer-Verlag, 1990.

[23] E. Jaulmes and A. Joux, *A Chosen-Ciphertext Attack against NTRU*, CRYPTO 2000, Lecture Notes in Computer Science 1880, pages 20-35, Springer-Verlag, 2000.

[24] C. Karimianpour, *Lattice-Based Cryptosystems*, Masters Thesis, University of Ottawa, 2008.

[25] R. Kouzmenko, *Generalizations of the NTRU Cryptosystem*, Diploma Project, École Polytechnique Federale de Lausanne, 2005-2006.

[26] A. K. Lenstra, H. W. Lenstra, and L. Lovasz, *Factoring Polynomials with Rational Coefficients*, Math. Ann. 261, pages 515-534, 1982.

[27] E. Malekian, A. Zakerolhosseini and A. Mashatan, *QTRU: A Lattice Attack Resistant Version of NTRU PKCS Based on Quaternion Algebra*, preprint, Available from: http://eprint.iacr.org/2009/386.pdf, Accessed: Dec. 2010.

[28] D. Micciancio and S. Goldwasser, *Complexity of Lattice Problems: A Cryptographic Perspective*, volume 671 of The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, Massachusetts, 2002.

[29] M. Nevins, C. Karimianpour and A. Miri, *NTRU over rings beyond $\mathbb{Z}$*, Designs, Codes and Cryptography, Volume 56, Number 1, pages 65-78, 2010.

[30] P. Q. Nguyen, *Hermite's Constant and Lattice Algorithms*, The LLL Algorithm: Survey's and Applications, pages 16-69, Information Security and Cryptography, Springer-Verlag, 2010.

[31] C. P. Schnorr, *A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms*, Theoretical Computer Science 53, pages 201-224, 1987.

[32] V. Shoup, *NTL: A Library for doing Number Theory*. http://www.shoup.net/ntl/. Accessed: Aug. 2010.

[33] J. H. Silverman, *Almost Inverses and Fast NTRU Key Creation*, NTRU Cryptosystems Technical Report 14, Version 1, 1999. Available from: http://www.ntru.com. Accessed: Dec 2010.

[34] J. H. Silverman, *High-Speed Multiplication of (Truncated) Polynomial Rings*, NTRU Cryptosystems Technical Report 11, 1999. Available from: http://www.ntru.com. Accessed: Dec 2010.