Angewandte Computer-
und Biowissenschaften

HOCHSCHULE
MITTWEIDA
University of Applied Sciences

HOCHSCHULE
MITTWEIDA
University of Applied Sciences

HOCHSCHULE
MITTWEIDA
University of Applied Sciences

HOCHSCHULE
MITTWEIDA
University of Applied Sciences

MASTER THESIS

Mr
Rasem Soufi

Lattice Based Cryptography
GGH & NTRU

2024

Faculty of **Applied Computer Sciences & Biosciences**

MASTER THESIS

Lattice Based Cryptography
GGH & NTRU

Author:
Rasem Soufi

Study Programme:
Applied Mathematics for Networks and Data sciences

Seminar Group:
MA21w1-M

First Referee:
Prof. Dr. rer. nat. Klaus Dohmen

Second Referee:
Prof. Dr. rer. nat. Peter Tittmann

Mittweida, Mai 2024

I. Preface

Developing computer processors and quantum computers requires continuous development of cryptographic algorithms. Researchers in standardization organizations keep experimenting with cryptographic algorithms and the related key sizes to recommend best practices for secure cryptography implementation [SE16]. The goal of researching Post Quantum Cryptography (PQC) is to develop cryptographic algorithms that are implemented in classic computers and secure against both quantum and classic computers [NIS16].

In this writing, we will have a primary introduction to lattices. Further, we will discuss the encryption and decryption functions of two lattice-based public key cryptography algorithms. The `Sagemath` codes are all experimented with using the `JupyterHub` server of the University of Applied Sciences Mittweida (Hochschule Mittweida).

Acknowledgment

First and forever, I am very grateful to God Almighty for His graces and blessings. I thank my small and big family to their endless support and encouragement.

I would love to specially thank

> Prof. Dohmen, for his support and guidance to achieve this work and much more other projects by his supervision.
>
> Prof. Tittmann, for his support, encouragement and guidance which I will be all of my life grateful for.

I would like to thank all my teachers (Prof. Viellmann, Prof. Baaske, Prof. Zaussinger, Dr. Nebel, Dr. Lange-Geisler, Mr. Stockmann and Mrs. Reader) for the priceless effort they were giving to afford me and my colleges lectures in the best quality. I would like to thank all the teaching and managerial staff of faculty of applied computer sciences & biosciences and of Hochschule Mittweida, you were all part of my chance to catch up again with my dreams.

To my Family; Rigerta, Ruba & Ryan.

HS-Mittweida, House 1, Spring 2024.

II. Contents

1 Preliminarities & Linear Algebra Notations

In this writing, we will denote horizontal vectors with bold small letters as

$$\mathbf{v} = (v_1, v_2, \cdots, v_n).$$

Matrices will be denoted by capital letters. Thus, we denote the multiplication of a matrix A times a vector $\mathbf{x}$ with $\mathbf{x}A$. Sets are also denoted by capital letters.

We will need the following rounding operation for $x \in \mathbb{R}$. The closest integer to x is defined as

$$\lfloor x \rceil = \lfloor x + \frac{1}{2} \rfloor \text{ [Reg04].}$$

We will consider the Euclidean norm

$$\|\mathbf{v}\| = \sqrt{v_1^2 + \cdots + v_n^2}.$$

Furthermore, we will use the Euclidean distance between two points

$$\mathrm{d}(\mathbf{v}, \mathbf{v}') = \|\mathbf{v} - \mathbf{v}'\| = \sqrt{\sum_{i=1}^{n} (v_i - v_i')^2}$$

1.1 Main Definitions

Definition 1.1 (cf. [Reg04]) Let $V = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$ be set of linearly independent vectors in $\mathbb{R}^n$. A *lattice* generated by these vectors $\mathscr{L}(\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n)$ is the set of all possible integer linear combinations of these vectors. For instance,

$$\mathscr{L}(V) = \left\{ \sum_{i=1}^{n} a_i \mathbf{v}_i \quad \text{where} \quad a_i \in \mathbb{Z} \right\}.$$

The number of vectors in V may be less than the dimension of the space. However, we will consider the so-called *full-rank* lattices in which the number of linearly independent vectors that generate the lattice equals the dimension of the space. The set V is called a *basis* of the lattice. In addition, we may represent the basis as an $n \times n$ matrix B in which the row vectors are the vectors of the basis V. Then we may denote the lattice to be generated by a set or a matrix

$$\mathscr{L}(V) = \mathscr{L}(B).$$

In case $\mathbf{v}_i \in \mathbb{Z}^n$ for all $i = 1, \cdots, n$, then $\mathscr{L}(B) \subseteq \mathbb{Z}^n$ and all the entries of any lattice vector are integers. This is called an *integral lattice*. In our writing, we will denote the full-rank integral lattice as simply a lattice. Furthermore, an element of $\mathscr{L}(B)$ may be denoted as a lattice vector or a lattice point.

Theorem 1.2 (cf. [Reg04]) *A lattice $\mathscr{L}(B) \subset \mathbb{R}^n$ equipped with vector addition forms a group.*

Proof: 1. Associativity is inherited from the vector space.

2. The closeness of addition is inherited from the closeness of integer addition. Because for two vectors $\mathbf{x_1}, \mathbf{x_2}$ in a lattice $\mathscr{L}(B)$ such that

$$\mathbf{x}_1 = a_1\mathbf{b}_1 + \cdots a_n\mathbf{b}_n$$
$$\mathbf{x}_2 = a'_1\mathbf{b}_1 + \cdots a'_n\mathbf{b}_n$$

where

$$a_1, \cdots, a_n, a'_1, \cdots, a'_n \in \mathbb{Z}$$

we obtain

$$\mathbf{x}_1 + \mathbf{x}_2 = (a_1 + a'_1)\mathbf{b}_1 + \cdots (a_n + a'_n)\mathbf{b}_n.$$

Since $(a_i + a'_i) \in \mathbb{Z}$ for all $i = 1, \cdots, n$, the vector $\mathbf{x}_1 + \mathbf{x}_2$ is an element in the lattice $\mathscr{L}(B)$.

3. The neutral element is simply the zero vector since

$$\mathbf{0}B = \mathbf{0} \in \mathscr{L}(B)$$

where

$$\mathbf{0} = (0, \cdots, 0) \in \mathbb{Z}^n.$$

4. The inverse element of any vector in the lattice $\mathbf{x} = \mathbf{a}B$ is $-\mathbf{x} = -\mathbf{a}B$ where $\mathbf{a}$ is an element in $\mathbb{Z}^n$.

□

1.1.1 several Bases of the same Lattice

The basis of a lattice is not unique. Several bases may generate the same lattice [Reg04].

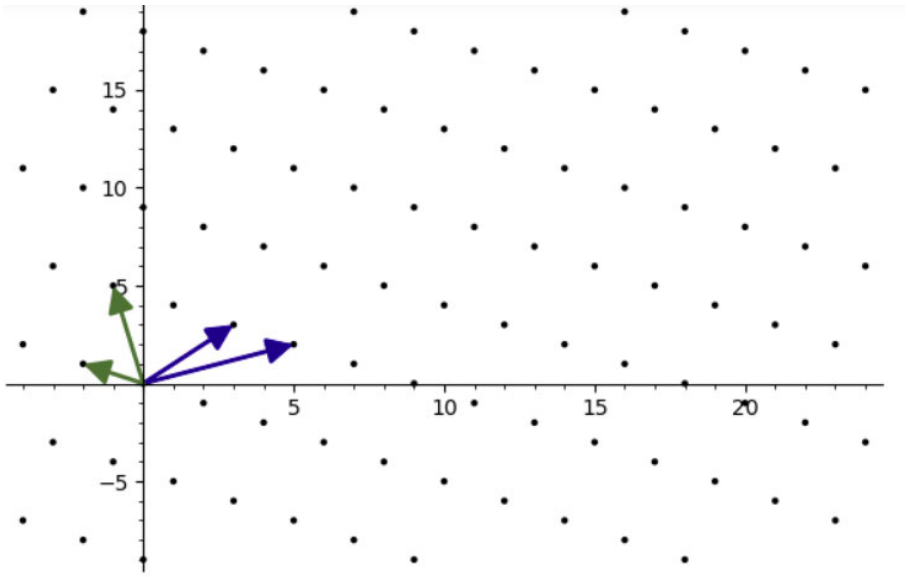

Figure 1.1: Lattice generated by the vectors $(5,2)$ and $(3,3)$.

Example 1.3 Consider the lattice in Figure 1.1. It is generated by the matrix

$$M = \begin{pmatrix} 5 & 2 \\ 3 & 3 \end{pmatrix}.$$

However, it may be generated by the matrix

$$M' = \begin{pmatrix} -1 & 5 \\ -2 & 1 \end{pmatrix}$$

Figure 1.1 is plotted using the following code in `Sagemath`.

```
def listOfPointsOfLattice(v_1,v_2,Xmin,Xmax,Ymin,Ymax):
    ListOfPoints = []
    I = 0
    for i in range(-100,100):
        for j in range(-100,100):
            I = (i*v_1) + (j*v_2)
            if(I[0]>Xmin and I[0] < Xmax and I[1] > Ymin and I[1] < Ymax):
                ListOfPoints.append(I)
    return(ListOfPoints)
```

The boundaries of the loops ranges in the previous routine are chosen arbitrarily. If the plots of the following statements are not as expected, these ranges may be changed.

```
Lattice_1 = point(listOfPointsOfLattice(vector([5,2]),vector([3,3]),
                                        -5,25,-10,20))
```

```
plot(Lattice_1)
```

We can generate the same points by running the following statements

```
Lattice_2 = point(listOfPointsOfLattice(vector([-1,5]), vector([-2,1]),
                                                    -5,25,-10,20))
plot(Lattice_2)
```

We can compare these two lists, however, we will change the data types to sets of elements to make the comparison easier. Thus we define the following routine to obtain a set of tuples from a list of points.

```
def Set_Tuples_from_list (ListofPoints):
    S = set()
    for i in range(len(ListofPoints)):
        K = (ListofPoints[i][0],ListofPoints[i][1])
        S.add(K)
    return(S)
```

Now we can compare two sets as follows.

```
S_1 = Set_Tuples_from_list(Points_1)
S_2 = Set_Tuples_from_list(Points_2)
S_1 == S_2
    True
```

Note that not any n linearly independent vectors in a lattice can generate the lattice [Reg04]. To clarify a first guess regarding this idea, we need the following definition.

Definition 1.4 (cf. [Reg04]) The *fundamental parallelepiped* of a lattice generated by basis B is

$$\mathscr{P}(B) = \{\mathbf{a}B \quad | \quad \mathbf{a} \in \mathbb{R}^n \quad \text{where} \quad 0 \leq a_i < 1\}.$$

Theorem 1.5 (cf. [Axl24]) *Let $V = \{\mathbf{v}_1, \cdots, \mathbf{v}_n\}$ be a basis of $\mathbb{R}^n$. The representation of any vector $\mathbf{x} \in \mathbb{R}^n$ as a linear combination of the vectors $\mathbf{v}_i$ is unique.*

Proof: By contradiction. Assume that we can represent a vector $\mathbf{x} \in \mathbb{R}^n$ as two different

linear combinations of the vectors in V. Thus,

$$\mathbf{x} = a_1\mathbf{v}_1 + \cdots + a_n\mathbf{v}_n$$
$$\mathbf{x} = a'_1\mathbf{v}_1 + \cdots + a'_n\mathbf{v}_n$$

with at least one $i \in \{1, \cdots, n\}$ such that $a_i \neq a'_i$. Then, by subtracting the previous two linear combinations we obtain

$$\mathbf{0} = (a_1 - a'_1)\mathbf{v}_1 + \cdots + (a_i - a'_i)\mathbf{v}_i + \cdots + (a_n - a'_n)\mathbf{v}_n$$

where at least the coefficient $(a_i - a'_i) \neq 0$. This is a nontrivial representation of the zero vector which contradicts that V is a basis. □

Theorem 1.6 (cf. [Reg04]) *An n linearly independent vectors in a lattice $L \subseteq \mathbb{Z}^n$ form a lattice basis B if and only if*

$$\mathscr{P}(B) \cap \mathscr{L}(B) = \{\mathbf{0}\}.$$

Proof: First, assume that B forms a basis of L. Since the coefficients of the vectors of B to obtain $\mathscr{P}(B)$ are in $[0, 1)$, the only possible lattice point in $\mathscr{P}(B)$ is $\{\mathbf{0}\}$.

Second, assume that

$$B = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\} \subset L$$

consists of linearly independent vectors with

$$\mathscr{P}(B) \cap L = \{\mathbf{0}\}.$$

The first conclusion is that any integer linear combination of the vectors $\mathbf{b}_1, \cdots, \mathbf{b}_n$ is an element in L by definition and closeness of vector addition in a lattice. Since $\mathbf{b}_1, \cdots, \mathbf{b}_n$ are linearly independent, then for all

$$\mathbf{x} \in L$$

there exists

$$a_1, \cdots, a_n \in \mathbb{R}$$

such that

$$\mathbf{x} = \sum_{i=1}^{n} a_i\mathbf{b}_i.$$

We want to prove that $a_1, \cdots, a_n$ are elements in $\mathbb{Z}$. Notice that

$$\sum_{i=1}^{n} \lfloor a_i \rfloor \mathbf{b}_i = \mathbf{x}'$$

is an element in L since it is an integer linear combination of the vectors of B. By

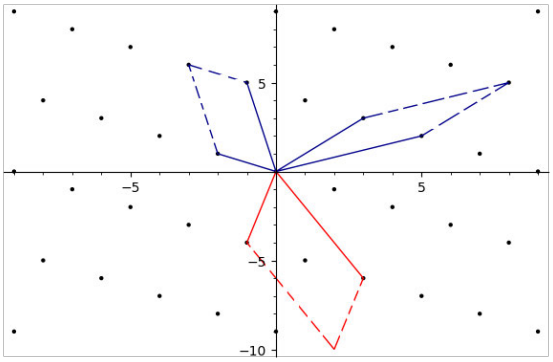

Figure 1.2: Fundamental parallelepipeds of several Bases.

closeness of addition of victors in L we conclude that

$$\mathbf{x}-\mathbf{x}' = \sum_{i=1}^{n}(a_i - \lfloor a_i \rfloor) \in L.$$

However,

$$(a_i - \lfloor a_i \rfloor) \in [0,1).$$

Thus,

$$\mathbf{x}-\mathbf{x}' = \mathbf{0}$$

and any vector in L can be obtained by an integer linear combination of the vectors of B.

□

In Example 1.3, consider the vectors sets

$$\begin{aligned} B_1 &= \{(5,2),(3,3)\},\\ B_2 &= \{(-1,5),(-2,1)\},\\ B_3 &= \{(3,-6),(-1,-4)\}. \end{aligned}$$

The lattice $\mathscr{L}(B_3)$ is a proper subset of the lattices $\mathscr{L}(B_1) = \mathscr{L}(B_2)$ since the lattice vector

$$(1,-5) = \frac{1}{2}(3,-6) + \frac{1}{2}(-1,-4) \in \mathscr{P}(B_3).$$

From Figure 1.2 we can conclude that a lattice vector is included in $\mathscr{P}(B_3)$. The lattice $\mathscr{L}(B_3)$ is called a *sublattice* of $\mathscr{L}(B_1) = \mathscr{L}(B_2)$ [Reg04].

With a second look at Figures (1.1,1.2) we will recognize that from each point of the lattice, we can copy $\mathscr{P}(B_1)$ and we will obtain a tiling of $\mathbb{R}^2$. In addition, we can obtain a tiling of $\mathbb{R}^2$ with copies of $\mathscr{P}(B_2)$.

Theorem 1.7 (cf [Reg04]) *If we take the translations $\mathscr{P}(B)+\mathbf{v}$ for all $\mathbf{v}$ in a lattice $L=\mathscr{L}(B)$ we will obtain a non-intersecting tiling of $\mathbb{R}^n$.*

For a vector $\mathbf{x}\in\mathbb{R}^n$, we have

$$\begin{aligned}\mathbf{x} &= \sum_{i=1}^{n}\alpha_i\mathbf{v}_i, \quad \text{such that} \quad \alpha_i\in\mathbb{R}\\ &= \sum_{i=1}^{n}(\alpha_i-\lfloor\alpha_i\rfloor)\mathbf{v}_i+\sum_{i=1}^{n}\lfloor\alpha_i\rfloor\mathbf{v}_i\\ &= \quad \mathbf{x}' \quad + \quad \mathbf{v}\end{aligned}$$

where

$$\mathbf{v}\in L, \quad \text{and} \quad \mathbf{x}'\in\mathscr{P}(B).$$

1.2 Unimodular Matrices

For our applications, we need several bases of the same lattice. However, plotting or checking if a lattice vector is included in the fundamental parallelepiped of a lattice is not always as easy as in our examples. Thus, a more sophisticated approach is necessary.

Definition 1.8 (cf. [Gil15]) A square matrix of integer entries $U\in\mathbb{Z}^{n\times n}$ is called *unimodular matrix* if $\det(U)=\pm1$.

Theorem 1.9 (cf. [Reg04]) *The inverse of an $n\times n$ unimodular matrix U is a unimodular matrix U^{-1}.*

Proof: First, we prove that the $\det U^{-1}=\pm1$. Note that $UU^{-1}=I$ where I is the identity $n\times n$ matrix. thus

$$\det(UU^{-1})=det(I).$$

Thus,

$$\det(U)\cdot\det(U^{-1})=1 \quad \Longrightarrow \quad \det(U^{-1})=\pm1.$$

Second, we prove that the entries of U^{-1} are integers. The inverse of an invertible matrix U can be calculated as follows

$$U^{-1}=\frac{1}{\det(U)}\operatorname{Adj}(U)=\pm\operatorname{Adj}(U) \text{ [Axl24].}$$

Note that $\operatorname{Adj}U$ denotes the adjoint matrix of U in which each element is a cofactor of

U. If we denote the elements of U^{-1} by u'_{ij} and the $ij-$minor of U by U_{ij}, then

$$u'_{ij} = (-1)^{i+j} \det(U_{ij})$$

which are integers. □

Since the following operations do not change the absolute value of the determinant of a matrix, we can apply them to any unimodular matrix and we will obtain another unimodular matrix.

Theorem 1.10 (cf. [Reg04]) *If any of the following operations are applied on a row vector of a unimodular matrix*

$$U = \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_n \end{pmatrix}$$

we will obtain another unimodular matrix. let $i, j \in \{1, \cdots, n\}$. The operations are

1. *Adding the scaler of a vector $\mathbf{u}_j$ to the vector $\mathbf{u}_i$ as*

$$\mathbf{u}_i \leftarrow \mathbf{u}_i + \alpha \mathbf{u}_j \quad \textit{where} \quad i \neq j \quad \textit{and} \quad \alpha \in \mathbb{Z}.$$

2. *Replace a vector with its inverse as*

$$\mathbf{u}_i \leftarrow -\mathbf{u}_i.$$

3. *Swapping two vectors as*

$$\mathbf{u}_i \leftrightarrow \mathbf{u}_j.$$

Theorem 1.11 ([Reg04]) *Two bases $B, B' \in \mathbb{Z}^{n \times n}$ generate the same lattice if and only if there exists a unimodular matrix $U \in \mathbb{Z}^{n \times n}$ such that $B' = UB$.*

Proof: First, assume that

$$\mathscr{L}(B') = \mathscr{L}(B) \tag{1.1}$$

and we want to prove that $B' = UB$ for some unimodular matrix U. From equation (1.1) we obtain that each row vector $\mathbf{b}'_i$ from B' can be written as an integer linear combination

of the row vectors of B as follows

$$\begin{aligned}
\mathbf{b}'_1 &= u_{11}\mathbf{b}_1 + u_{12}\mathbf{b}_2 + \cdots + u_{1n}\mathbf{b}_n \\
\mathbf{b}'_2 &= u_{21}\mathbf{b}_1 + u_{22}\mathbf{b}_2 + \cdots + u_{2n}\mathbf{b}_n \\
&\vdots \\
\mathbf{b}'_n &= u_{n1}\mathbf{b}_1 + u_{n2}\mathbf{b}_2 + \cdots + u_{nn}\mathbf{b}_n.
\end{aligned}$$

Now we can write the previous system of linear equations as a matrix multiplication

$$UB = \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ u_{21} & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{nn} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_n \end{pmatrix} = B'$$

In the same manner, we can write the matrix $B = VB'$ where $V \in \mathbb{Z}^{n\times n}$. What is left is to prove that $\det(U) = \pm 1$ and $\det(V) = \pm 1$. We can write

$$B = VB' = VUB.$$

Thus

$$BB^{\mathsf{T}} = VUBB^{\mathsf{T}}(VU)^{\mathsf{T}} \text{ [Reg04].}$$

Then

$$\det(BB^{\mathsf{T}}) = \det(BB^{\mathsf{T}})(\det(VU))^2$$

and

$$\det(V)\det(U) = \pm 1.$$

We know that the determinant of a matrix is a combination of additions and subtractions of products of the elements of the matrix and since U and V are of integer elements, we conclude that $\det(U) = \pm 1$ and $\det(V) = \pm 1$. Thus, U and V are unimodular matrices.

Now we prove the other direction. Precisely, assuming that $B' = UB$ where U is a unimodular matrix and $B, B' \in \mathbb{R}^{n\times n}$, we will prove that $\mathscr{L}(B') = \mathscr{L}(B)$. By this assumption, we conclude that each row in B' is an integer linear combination in the rows of B. Thus, each row in B' is contained in $\mathscr{L}(B)$. We obtain

$$\mathscr{L}(B') \subseteq \mathscr{L}(B). \tag{1.2}$$

In addition, we have $B = U^{-1}B'$. Since U^{-1} is a unimodular matrix (Theorem 1.9), we conclude that

$$\mathscr{L}(B) \subseteq \mathscr{L}(B') \tag{1.3}$$

considering arguments of relation (1.2). From relations (1.2,1.3) we have $\mathscr{L}(B) = \mathscr{L}(B')$ □

In the specific case of a full-rank lattice, we can define the volume of the fundamental parallelepiped as a variant of the lattice.

Definition 1.12 (cf [Reg04]) let $L = \mathscr{L}(B)$ be a lattice and $P = \mathscr{P}(L)$ be its fundamental parallelepiped. The *volume of the fundamental parallelepiped* is defined as $\mathrm{Vol}(\mathscr{P}(L)) = |\det(B)|$. It may also be denoted as $\det(L)$.

We know that for two different bases B, B' of the same lattice, there exists a unimodular matrix U such that $B' = U \cdot B$. Thus,

$$\begin{aligned}\det(B') &= \det(U \cdot B)\\ &= \det(U) \cdot \det(B)\\ &= \pm\det(B).\end{aligned}$$

By taking the absolute value to both sides we obtain

$$|\det(B')| = |\det(B)|.$$

Consequently, $\mathrm{Vol}(\mathscr{P}(\mathscr{L}(B)))$ is independent of the chosen basis.

In case the lattice L is a subset of $\mathbb{Z}^n$ with dimension

$$\dim L = m < n.$$

Thus, if M the matrix of a set of linear independent vectors that generate L, then,

$$M \in \mathbb{Z}^{m \times n}.$$

In this case

$$\mathrm{Vol}(\mathscr{P}(L)) = |\sqrt{\det(M^{\mathsf{T}}M)}|.$$

Theorem 1.13 *Let M be a matrix of n linearly independent vectors of a lattice L of dimension n. If*

$$|\det(M)| = \mathrm{Vol}(\mathscr{P}(L)), \tag{1.4}$$

then

$$\mathscr{L}(M) = L.$$

Proof: Note that if the lattice $\mathscr{L}(M)$ does not equal L, then there exists a lattice point in the intersection of L with the parallelepiped of the vectors of M denoted $\mathscr{P}(M)$. This contradicts the volume of the fundamental parallelepiped of L in relation 1.4. □

The following theorem is a direct conclusion of the previous one. Because the included

operations give us a full rank matrix M' from an original one M with

$$\det(M) = \det(M').$$

Theorem 1.14 *Two bases generate the same lattice if and only if one is obtained from the other by applying finite number of the following operations on the row vectors.*

1. $\mathbf{b}_i \leftarrow \mathbf{b}_i + k\mathbf{b_j}$.
2. $\mathbf{b}_i \leftrightarrow \mathbf{b}_j$.
3. $\mathbf{b}_i \leftarrow -\mathbf{b}_i$.

Note that $\mathbf{b}_i \neq \mathbf{b}_j$ *and* $k \in \mathbb{Z}$.

Example 1.15 Let $B = \{(1,1),(-1,1)]\}$. Note that by taking the matrix

$$A = \begin{pmatrix} 25 & 7 \\ 7 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 18 & 32 \\ 5 & 9 \end{pmatrix}$$

we conclude that $\mathscr{L}(B) = \mathscr{L}(A)$

Example 1.16 In Example(1.3) we may solve for X the matrix equation

$$XM = M'$$
$$X \cdot \begin{pmatrix} 5 & 2 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} -1 & 5 \\ -2 & 1 \end{pmatrix}.$$

We obtain

$$X = \begin{pmatrix} -2 & 3 \\ -1 & 1 \end{pmatrix}$$

which is a unimodular matrix. Note that if we start from the other matrix as $X'M' = M$ we obtain

$$X' = \begin{pmatrix} 1 & -3 \\ 1 & -2 \end{pmatrix}$$

which is unimodular and the inverse of X.

1.2.1 Software Implementation

We will check what are the possible results of implimenting the previously described ideas using `Sagemath`. As a first step, we define the inner product of two vectors.

```
def InnerProduct (v_1,v_2):
    s = 0
    for i in range(len(v_1)):
        s+= v_1[i] * v_2[i]
    return(s)
```

Depending on this routine, we define the Euclidean length

```
def EuclideanLength(v):
    return(RR(sqrt(InnerProduct(v,v))))
```

The *cosine* of the angle θ between two vectors $v_1, v_2 \in V$ where V is a vector space can be defined as follows [Axl24]

$$cos(\theta) = \frac{< v_1, v_2 >}{\|v_1\| \cdot \|v_2\|}.$$

We can define this in `Sagemath` as follows

```
def cosinus (v_1,v_2):
    if (EuclideanLength(v_1) != 0 and EuclideanLength(v_2) != 0):
        x = InnerProduct(v_1,v_2) / (EuclideanLength(v_1)*EuclideanLength(v_2))
        return(x)
    else:
        print("The Function is not possible")
```

Now we can check for our lattice bases in Example (1.15)

```
cosinus([1,1],[-1,1])
    0.000000000000000
cosinus([18,32],[5,9])
    0.999986002925388
```

In other words, we could have obtained one orthogonal basis and another basis with extremely small angle between its vectors where both generate the same lattice.

1.3 Hadamard's Ratio

Since we will consider lattices of dimensions much bigger than 2, The approach to check the orthogonality of a lattice basis which we have just applied is not applicable. Thus we will define a better approach.

Definition 1.17 (cf. [Gil15]) *Hadamard's ratio* of a lattice basis

$$B = \{\mathbf{v}_1, \cdots, \mathbf{v}_n\} \subset \mathbb{Z}^n$$

is defined as follows

$$\mathscr{H}(B) = \left(\frac{\mathrm{Vol}(\mathscr{P}(\mathscr{L}(B)))}{\|\mathbf{v}_1\| \cdots \|\mathbf{v}_n\|}\right)^{\frac{1}{n}} = \left(\frac{|\det(B)|}{\|\mathbf{v}_1\| \cdots \|\mathbf{v}_n\|}\right)^{\frac{1}{n}}.$$

The Hadamard's ratio is a function in the vectors of the chosen basis of the lattice. Note that

$$0 < \mathscr{H}(B) \leq 1.$$

If $\mathscr{H}(B)$ is close to 1, then the vectors of B are called *reasonably orthogonal* and the related basis is a *good basis*. However, if $\mathscr{H}(B)$ is close to 0 then there are several small angles between the vectors of the basis and thus they are called *reasonably parallel* and the related basis is a *bad basis*.

Example 1.18 In Example 1.15 we may check

$$\mathscr{H}(B) = \frac{2}{2} = 1$$

which tells that B is a good basis. However,

$$\mathscr{H}(A) = 0.005$$

which tells us that A is a bad basis.

In `Sagemath`, we can define Hadamard's ratio as a function of matrices.

```
def Hadamard (M):
    v = 1
    for i in M:
        v *= EuclideanLength(i)
    return((abs(M.det())/v).nth_root(M.nrows()))
```

1.4 Lattice Hard Problems

To build cryptographic systems, computationally hard problems are used as building blocks (Cryptoprimitives) as factorization in RSA or discrete logarithms in ElGamal cryptosystem and Deffie Hellmann Key exchange [Gil15]). The following lattice-related prob-

lems are considered unsolved [Jef14].

1. *The Closest Vector Problem* (CVP): For a given lattice L and a vector

$$\mathbf{v} \in \mathbb{R}^n \setminus L,$$

find the closest lattice vector to $\mathbf{v}$. In other words find $\mathbf{w} \in L$ such that

$$\|\mathbf{v}-\mathbf{w}\| \leq \|\mathbf{v}-\mathbf{w}'\|, \forall \mathbf{w}' \in L.$$

2. *The Shortest Vector Problem* (SVP): Find a shortest non-zero vector in a given lattice L. This is a special case of the CVP by considering the closest vector to the zero vector. Note that we use the term "a" shortest vector instead of "the" shortest vector, because there exists several shortest vectors in a lattice. For instance, if $\mathbf{v}$ is a shortest vector in L, then $-\mathbf{v}$ is another shortest vector in L.

3. *Approximate Shortest Vector Problem* (ASVP) Assume a function $\phi(n)$. Let L be a lattice of dimension n. Let $\mathbf{w}$ be a shortest vector in L. Find a lattice vector $\mathbf{v}$ such that

$$\|\mathbf{v}\| \leq \phi(n)\|\mathbf{w}\|.$$

For example, one may search for a lattice vector that satisfy

$$\|\mathbf{v}\| \leq \sqrt{n}\|\mathbf{w}\|,$$

or

$$\|\mathbf{v}\| \leq 2^{n/4}\|\mathbf{w}\|.$$

If one solves the SVP then ASVP can be solved too. However in some applications, we depend on the hardness of ASVP instead of SVP[1].

1.5 Rounding Method

Let L be a lattice generated by any of two bases $\mathscr{L}(B) = \mathscr{L}(B') = L$. Let $\mathscr{H}(B) \approx 1$ and $\mathscr{H}(B') \approx 0$. Let $\mathbf{x} \in \mathbb{R}^n$ with representation

$$\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{b}_i = \sum_{i=1}^{n} x'_i \mathbf{b}'_i.$$

By rounding the coefficients in previous combinations we obtain

$$\mathbf{x}_1 = \sum_{i=1}^{n} \lfloor x_i \rceil \mathbf{b}_i \quad \text{and} \quad \mathbf{x}_2 = \sum_{i=1}^{n} \lfloor x'_i \rceil \mathbf{b}'_i.$$

[1] There are several other variants of lattice unsolved problems as approximate closest vector problem or the shortest basis problem. However, these are not included here because they are not related to our applications.

Clearly, $\mathbf{x}_1, \mathbf{x}_2 \in L$. L. Babai has introduced this rounding method in [Bab85] to obtain the closest lattice point to a target point depending on a good basis. Let $\mathbf{y}$ be the closest lattice point to our target point $\mathbf{x}$. In the same paper, he introduced a factor of error

$$|\mathbf{x} - \mathbf{x}_1| \leq |\mathbf{x} - \mathbf{y}| \cdot \left(1 + 2n \cdot (9/2)^{n/2}\right)$$

assuming the $sin(\theta) \geq (\sqrt{2}/3)$ where θ is the angle between any basis vector $\mathbf{b}_i$ and the subspace $\text{span}(B \setminus \{\mathbf{b}_i\})$.

A further demonstration was introduced in [Jef14] to demonstrate that applying the rounding method on a vector $\mathbf{x}$ will return a vertex of the copy of the fundamental parallelepiped containing $\mathbf{x}$. When the basis is bad (Figure1.3), the returned vertex will be far away from the correct one which can be reached when the basis is good (Figure 1.4).

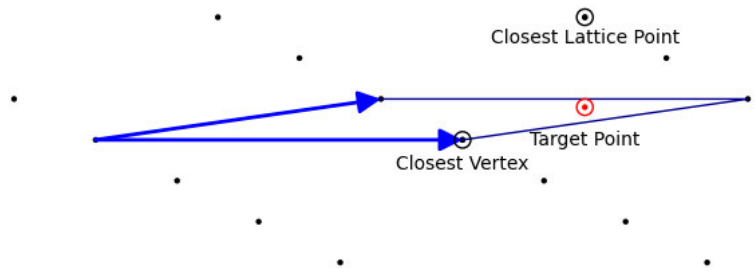

Figure 1.3: Rounding-off Procedure returns a wrong lattice point if the given basis is bad.

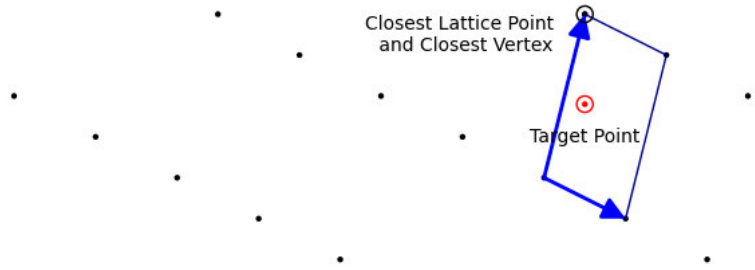

Figure 1.4: Rounding-off Procedure returns the closest lattice point if the basis is reasonably orthogonal.

Since $\mathscr{H}(B) \approx 1$ We conclude that $|\mathbf{x}-\mathbf{x}_1| < |\mathbf{x}-\mathbf{x}_2|$. Moreover, $|\mathbf{x}-\mathbf{x}_1| \leq |\mathbf{x}-\mathbf{z}|$ for all $\mathbf{z} \in L$.

Example 1.19 Lets take the Lattice $L = \mathscr{L}(B)$ which is generated by the matrix

$$B = \begin{pmatrix} -30 & 0 & 20 \\ 20 & 10 & 30 \\ 20 & -130 & 30 \end{pmatrix}.$$

By using the unimodular matrix

$$U = \begin{pmatrix} 10 & 39 & 131 \\ 10 & 40 & 133 \\ 3 & 12 & 40 \end{pmatrix}$$

we obtain

$$B' = U \cdot B = \begin{pmatrix} 3100 & -16640 & 5300 \\ 3160 & -16890 & 5390 \\ 950 & -5080 & 1620 \end{pmatrix}$$

for which $L = \mathscr{L}(B')$. Note that $\mathscr{H}(B) = 1$ while $\mathscr{H}(B') \approx 1.052 \cdot 10^{-7}$.

Consider the point $\mathbf{p} = (100, 100, 100) \notin L$. We want to find the closest lattice point $\mathbf{p}' = \mathbf{x} \cdot B$ to $\mathbf{p}'$. We apply Babai's algorithm. First, we need the solution

$$\mathbf{p} = (-\frac{10}{13}, \frac{30}{7}, -\frac{40}{91}) \cdot B.$$

By taking $\lfloor x_i \rceil$ for all the entries of the coefficients vector in the previous representation of $\mathbf{p}$ we obtain

$$\mathbf{p}' = (-1, 4, 0) \cdot B = (110, 40, 100)$$

If we apply the same steps considering B' we obtain

$$\begin{aligned} \mathbf{p} &= (-\frac{670}{91}, \frac{2010}{91}, -\frac{4490}{91})B' \\ \mathbf{p}'' &= (-7, 22, -49)B' \\ &= (1270, -6180, 2100) \end{aligned}$$

For sure both $\mathbf{p}'$ and $\mathbf{p}''$ are lattice points, however,

$$\mathrm{d}(\mathbf{p}, \mathbf{p}') \approx 60.827 \ll 6693.825 \approx \mathrm{d}(\mathbf{p}, \mathbf{p}'').$$

Note that from our example and figures, the CVP seems to be an easy problem. However, it gets exponentially harder with respect to the dimensionality of the lattice [Gil15].

In the rounding method, if the dimension of the lattice is n, then the fundamental parallelepiped contains 2^n vertices. Note that the hardness of CVP is resistant to attacks that are designed by quantum computers. In other words, there is still no efficient cryptoanalytic algorithm that works in quantum computers or classic computers to solve CVP efficiently [Gil15].

1.6 Lattice Shortest Vector Estimation

Finding a shortest vector or its length are, in general, open problems. For some applications, we are interested in estimating the length of a shortest vector in a given Lattice L without the knowledge of any good basis. There were several investigations to solve this problem. These investigations were searching for the shortest length in a lattice with respect to the dimension of the lattice n and the volume of its fundamental parallelepiped $\det(L)$. According to [Jef14], the best estimation is the *Gaussian expected shortest length* which is defined as

$$\sigma(L) = \sqrt{\frac{n}{2\pi e}}(\det(L))^{\frac{1}{n}}. \tag{1.5}$$

Precisely, this result tells that a shortest non-zero vector $\mathbf{v}$ in a lattice will satisfy

$$\|\mathbf{v}\| \approx \sigma(L).$$

This estimation is supposed to give fine results in high dimensions. It is derived from the Hermit's Theorem.

Theorem 1.20 (Hermit [Jef14]) *Every lattice L of dimension n contains a non-zero vector $\mathbf{v}$ satisfying*

$$\|\mathbf{v}\| \leq (\det(L))^{\frac{1}{n}}.$$

Hermit's approximation is itself is a refinement of Minikowski's Theorem.

Theorem 1.21 (Minikowski [Jef14]) *Let $S \subset \mathbb{R}^n$ be a bounded symmetric convex set. Let L be a lattice in $\mathbb{R}^n$. If*

$$\mathrm{Vol}\, S > 2^n \det(L)$$

then S contains a non-zero lattice vector. If S is additionally closed, then it is sufficient to take

$$\mathrm{Vol}\, S \geq 2^n \det(L).$$

We will not dive into the proves of these theorems, however, we will have an illustration

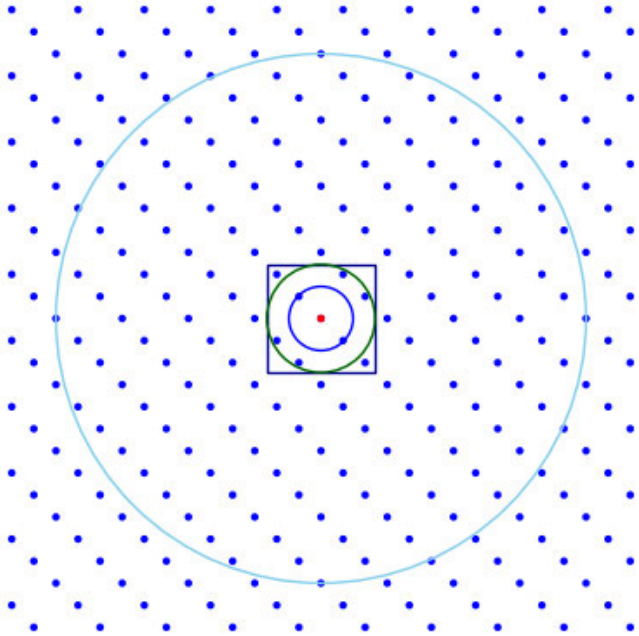

Figure 1.5: Estimation of the shortest length in a lattice with respect to n and $\det(L)$.

in Figure 1.5 where we have

- The big circle represents Minikowski's Theorem.
- The square represents Hermit's Theorem. The proof of this theorem depends on the hypercube

  $$S = \{(x_1, \cdots, x_n) \in \mathbb{R}^n \ : \ -(\det(L))^{\frac{1}{n}} \leq x_i \leq \det(L)^{\frac{1}{n}} \ \forall \, i = 1, \cdots, n\}$$

  which is a closed-bounded symmetric convex set.
- The small green circle represents the Gaussian expected shortest length.
- The smallest circle represents the actual shortest length.
- The red point is the zero vector.

1.6.1 A Proof of Rounding Method in Lattices with Orthogonal Bases

Let $\mathbf{p}$ be a lattice point generated using an arbitrary basis. Let $\mathbf{p}' = \mathbf{p} + \mathbf{r}$ where $\mathbf{r}$ is an error vector with a special range of length such that $\mathbf{p}$ is the closest lattice point to $\mathbf{p}'$. Using an orthogonal basis we may obtain $\mathbf{p}$ from $\mathbf{p}'$.

Theorem 1.22 *Let $L = \mathscr{L}(B)$ be a lattice generated by an orthogonal basis B. The*

shortest vector in L is the shortest vector in B.

Proof: Let $\mathbf{v} \in L$ and let $\mathbf{b} \in B$ be the shortest vector in B. We want to prove that

$$\|\mathbf{v}\| \geq \|\mathbf{b}\|. \tag{1.6}$$

This vector $\mathbf{v}$ is in one of the following cases

1. It may be a vector in B which will lead to inequality (1.6)
2. It may be a linear combination of vectors of B with coefficients $\{-1, 0, 1\}$ which by Pythagoras theorem [Axl24] will lead to inequality (1.6).
3. If there exists a coefficient α in the linear combination of $\mathbf{v}$ with $|\alpha| > 1$, we also conclude inequality (1.6) by Pythagoras theorem and by $\|\mathbf{b}\| < |\alpha| \cdot \|\mathbf{b}\|$.

□

Theorem 1.23 *Let $\mathbf{p}, \mathbf{p}'$ be two different points from the lattice $L = \mathscr{L}(B)$ where B is an orthogonal basis with shortest vector $\mathbf{b}_k$, thus*

$$\|\mathbf{b}_k\| \leq \|\mathbf{b}_i\|, \quad \forall \mathbf{b}_i \in B. \tag{1.7}$$

Then the distance between $\mathbf{p}$ and $\mathbf{p}'$ is

$$\|\mathbf{p} - \mathbf{p}'\| \geq \|\mathbf{b}_k\|.$$

Proof: We know that

$$\|\mathbf{p} - \mathbf{p}'\| = \sqrt{\sum_{i=1}^{n} (a_i - a_i')^2 \|\mathbf{b}_i\|^2}$$

where

$$\mathbf{p} = \sum_{i=1}^{n} a_i \mathbf{b}_i, \quad \text{and} \quad \mathbf{p}' = \sum_{i=1}^{n} a_i' \mathbf{b}_i.$$

From relation (1.7) we conclude

$$\|\mathbf{p} - \mathbf{p}'\| \geq \sqrt{\sum_{i=1}^{n} (a_i - a_i')^2 \|\mathbf{b}_k\|^2}.$$

The term $(a_i - a_i')^2$ can be minimized when $a_i = a_i'$ for all $i = 1, \cdots, n$. However since $\mathbf{p} \neq \mathbf{p}'$ there must exist $l \in \{1, \cdots, n\}$ for which

$$(a_l - a_l')^2 \neq 0. \tag{1.8}$$

Since $a_i, a_i' \in \mathbb{Z}$ then $\min(a_l - a_l')^2 = 1$ and thus

$$\|\mathbf{p} - \mathbf{p}'\| \geq \|\mathbf{b}_k\|.$$

□

Theorem 1.24 *Let $B = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$ be an orthogonal basis of the lattice $L = \mathscr{L}(B)$. Let $\mathbf{b}_k$ be the shortest vector in B. Thus,*

$$\|\mathbf{b}_k\| \leq \|\mathbf{b}_i\|, \qquad \forall i = 1, \cdots, n. \tag{1.9}$$

Let X be the open ball with a radius equal to half of the shortest vector in B. Thus

$$X = B(\mathbf{0}, \frac{1}{2}\|\mathbf{b}_k\|).$$

Then, for any vector $\mathbf{v} \in X$ where $\|\mathbf{v}\| < \frac{1}{2}\|\mathbf{b}_k\|$, the linear combination

$$\mathbf{v} = \sum_{i=1}^{n} a_i \mathbf{b}_i$$

consists of coefficients $|a_i| < 1/2$ for all $i = 1, \cdots, n$.

Proof: By Pythagoras Theorem we obtain that

$$\|a_i \mathbf{b}_i\| < \frac{1}{2}\|\mathbf{b}_k\|, \quad \forall i = 1, \cdots, n. \tag{1.10}$$

Thus, from inequalities (1.9,1.10) we obtain

$$|a_i| \cdot \|\mathbf{b}_k\| \leq |a_i| \cdot \|\mathbf{b}_i\| < \frac{1}{2}\|\mathbf{b}_k\|$$

Then, $|a_i| < 1/2$ for all $i = 1, \cdots n$. □

Conclusion 1 *Considering the assumptions of Theorem 1.24. By rounding the coefficients of the vector $\mathbf{v}$ we obtain*

$$\mathbf{v}' = \sum_{i=1}^{n} \lfloor a_i \rceil \mathbf{b}_i = \mathbf{0}.$$

Theorem 1.25 *Let $B = \{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$ be an orthogonal basis of the lattice $L = \mathscr{L}(B)$. Let $\mathbf{b}_k$ be the shortest vector in B. Let*

$$X = \bigcup_{\mathbf{u} \in L} B(\mathbf{u}, \frac{1}{2}\|\mathbf{b}_k\|).$$

Then,

$$\forall \mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{b}_i \in X \subset \mathbb{R}^n$$

the vector

$$\mathbf{x}' = \sum_{i=1}^{n} \lfloor x_i \rceil \mathbf{b}_i = \mathbf{u}.$$

In other words, rounding the coefficients of the representation of any vector in the open ball $B(\mathbf{u}, \frac{1}{2}\|\mathbf{b}_k\|)$ concerning B will return $\mathbf{u}$ which is the center of the ball.

Proof: Let

$$\mathbf{u} = \sum_{i=1}^{n} u_i \mathbf{b}_i.$$

Note that $\mathbf{x} = \mathbf{u} + \mathbf{v}$ for which $\|\mathbf{v}\| < \frac{1}{2}\|\mathbf{b}_k\|$. Thus, if

$$\mathbf{v} = \sum_{i=1}^{n} a_i \mathbf{b}_i \quad \text{then,} \quad |a_i| < \frac{1}{2}, \quad \forall i = 1, \cdots, n \tag{1.11}$$

according to theorem 1.24. We know that,

$$\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{b}_i = \sum_{i=1}^{n} (u_i + a_i) \mathbf{b}_i.$$

From the relation (1.11) we obtain $-1/2 < a_i < 1/2$. Thus,

$$u_i < u_i + a_i + \frac{1}{2} = x_i + \frac{1}{2} < u_i + 1.$$

We obtain $\lfloor x_i \rceil = u_i$. □

1.7 Lattice Reduction

Finding a good basis of a given lattice is called *lattice reduction* and a good basis of a given lattice is called a *reduced basis*. A reduced basis can easily solve the lattice hard problems (SVP and CVP) [Jef14].

Lattice reduction is known to be a mathematical hard problem. In other words, currently, no effecient algorithm exists to solve this problem (especially in the case of high dimensional lattices as we will discuss in cryptographic applications) [Jef14]. However, there are some algorithms that reduce lattices with low dimensions.

1.7.1 Gauss Lattice Reduction Algorithm

The first algorithm was given by Gauss. It reduces lattices of dimension 2. According to [Jef14], the algorithm is implemented as follows.

Let $L = \mathscr{L}(\{\mathbf{v}_1, \mathbf{v}_2\})$. Repeat the following steps until $m = 0$ in step (*b*).

Algorithm 1

(a) If $\|\mathbf{v}_1\| > \|\mathbf{v}_2\|$ *swap the two vectors to obtain* $\|\mathbf{v}_1\| \leq \|\mathbf{v}_2\|$.

(b) Reduce $\mathbf{v}_2$ *by replacing it with*

$$\mathbf{v}_2^* = \mathbf{v}_2 - m\mathbf{v}_1 \text{ where } m = \left\lfloor \frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_2\|^2} \right\rceil .$$

Note that if we don't consider the rounding to the closest integer process in step *(b)* in the previous algorithm, then $\mathbf{v}_2^*$ is the projection of $\mathbf{v}_2$ on the orthogonal complement of $\mathbf{v}_1$. However, this may not be a lattice vector [Jef14].

To implement this method in `Sagemath` we may apply the following routine

```
def Gauss_Reduced_Lattice (M):
    m =1
    while (m!=0):
        if (EuclideanLength(vector(M[0]))>EuclideanLength(vector(M[1]))):
            M = matrix([list(M[1]),list(M[0])])
        #print(M)
        m = round(N(InnerProduct(M[0],M[1]))/(EuclideanLength(M[0]))^2)
        #print(m)
        X = M[1] - (m*M[0])
        M = matrix([M[0],X])
    return(M)
```

Let's try the previous code to obtain a good basis from the basis

$$B = \begin{pmatrix} 10131 & 62742 \\ 71243 & 441213 \end{pmatrix}.$$

The final result is

$$B' = \begin{pmatrix} 20 & 3 \\ 1 & 30 \end{pmatrix}$$

If we keep the out-commented statemenats, we will see the steps that the Gauss algorithm took to reach this result as described in Table 1.1.

	$\mathbf{v}_1$	$\mathbf{v}_2$	m
1	$(10131, 62742)$	$(71243, 441213)$	7
2	$(326, 2019)$	$(10131, 62742)$	31
3	$(25, 153)$	$(326, 2019)$	13
4	$(1, 30)$	$(25, 153)$	5
5	$(20, 3)$	$(1, 30)$	0

Table 1.1: Gauss Lattice Reduction Algorithm Example.

1.7.2 LLL Lattice Reduction Algorithm

In this algorithm, we need to implement Gram-Schmidt orthogonalization. For a given basis $\{\mathbf{v}_1, \cdots, \mathbf{v}_n\}$ of a vector space V, the Gram-Schmidt orthogonal basis $\{\mathbf{v}_1^*, \cdots, \mathbf{v}_n^*\}$ is given as follows [Jef14].

Algorithm 2

(a) Let $\mathbf{v}_1^* = \mathbf{v}_1$.

(b) For $i = 1, \cdots, n$ *we let*

$$\mathbf{v}_i^* = \mathbf{v}_i - \sum_{j=1}^{i-1} \frac{\mathbf{v}_i \mathbf{v}_j^*}{\|\mathbf{v}_j^*\|^2} \mathbf{v}_j^*.$$

Gram-Schmidt algorithm gives an orthogonal basis of the vector space. However, if the vectors $\mathbf{v}_1, \cdots, \mathbf{v}_n$ form a basis of a lattice L, Gram-Schmidt algorithm does not necessary return a basis of L. Because it contains linear combinations of the vectors in the original basis with non-integer coefficients [Jef14]. We will denote the Gramm-Schmidt coefficients as

$$\mu_{i,j} = \frac{\mathbf{v}_i \cdot \mathbf{v}_j^*}{\|\mathbf{v}_j^*\|^2} \text{ for } 1 \leq j \leq i-1.$$

The LLL-algorithm was founded by A.K. Lenstra, H.W. Lenstra Jr, L. Lovász in 1982 [Jef14]. The main objective is to reduce a lattice basis into an LLL-reduced basis [Jef14]. We may define an *LLL-reduced basis* to be a lattice basis that fulfills the following conditions [Jef14].

1. The size condition:

$$|\mu_{i,j}| \leq \frac{1}{2} \text{ for all } 1 \leq j < i \leq n.$$

2. Lovász Condition:

$$\|\mathbf{v}_i^*\|^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right) \|\mathbf{v}_{i-1}^*\|^2 \text{ for all } 1 < i \leq n.$$

Considering Gramm-Schmidt reduction and assuming the vectors $\mathbf{v}_1, \cdots, \mathbf{v}_n$ form a basis of lattice L, the LLL-algorithm is as follows [Jef14].

Algorithm 3

(a) Set $\mathbf{v}_1^* = \mathbf{v}_1$.

(b) Set $k = 2$. *In the following steps, we will consider* $\mathbf{v}_k$ *to be the current working vector.*

(c) If $k < n+1$,

1. *Assign a new vector to* $\mathbf{v}_k$ *as*

$$\mathbf{v}_k \to \mathbf{v}_k - \sum_{j=1}^{k-1} \lfloor \mu_{k,j} \rceil \mathbf{v}_j.$$

This step needs the calculation of Gramm-Scmidt vectors $\mathbf{v}_1^*, \cdots, \mathbf{v}_k^*$.

2. *If*

$$\|\mathbf{v}_k^*\|^2 \geq \left(\frac{3}{4} - \mu_{k,k-1}^2\right) \|\mathbf{v}_{k-1}\|^2,$$

then assign to k *the value* $k+1$ *and repeat from step (c). Otherwise, swap*

$$\mathbf{v}_{k-1} \leftrightarrow \mathbf{v}_k,$$

and set $k = \max(k-1, 2)$ *and repeat from step (c).*

(d) Output the vectors $\{\mathbf{v}_1, \cdots, \mathbf{v}_n\}$.

According to Theorem 1.14 and since the main loop in the LLL-algorithm adds an integer linear combination of basis vectors to another basis vector and it may swap vectors, we know that the output is a basis of the same lattice L. In fact, what LLL-algorithm does is reducing the volume of the fundamental domains of the sublattices without changing the volume of the fundamental domain of the lattice L [Jef14]. The Lovász condition will guarantee that the lengths of the basis vectors are gradually growing from $\mathbf{v}_1$ to $\mathbf{v}_n$ [Jef14]. Furthermore, there are several results regarding the independence of the goodness of the output from the parameter $3/4$. However, the output is dependent on the order of the row vectors of the given basis [Jef14].

If we combine the LLL-algorithm with Babai's rounding method, we will have a good approximation of the closest vector problem. This combination is done by applying LLL- Algorithm and then Babai's rounding method [Gil15]. This was the method to break some lattice-based cryptosystems [Jef14]. However, for lattices with high dimensions where $n \geq 500$, LLL-algorithm is not feasible [Jef14]. In the following Chapter, we will discuss two cryptosystems and we will use LLL-algorithm in one Cryptoanalysis method.

To implement LLL-algorithm, we will use the *Sagemath* Built-in routine. It takes a matrix and returns the related LLL matrix. We will use the matrix B' in Example 1.19.

```
M = matrix([[-30,1,20], [20,10,29],[20,-131,30]])
print(M.lll())
```

The output is exactly the matrix B from the same example.

2 Lattice Based Public Key Cryptosystems

Cryptographers search for mathematical hard problems with trapdoors to be the building blocks, or the so-called crypto primitives, in the design of cryptosystems [Gil15]. In the following, we will discuss a cryptosystem that depends on CVP and another cryptosystem which has a security proof even against lattice based attacks.

2.1 GGH

Named after Oded Goldreich, Shafi Goldwasser, and Shai Halevi who designed this system [GGH97]. Since Phong Nguyan has proven some weaknesses in it, this system didn't get developed for standardization [Ngu99]. There are some improvements to the system [AA20]. However, it does not have a security proof yet [Jef14]. However, it plays the role of an intuitive design that implements CVP in cryptography [Gil15].

In the following, Alice is the receiver, Bob is the sender, and Eve is the eavesdropper who is trying to decrypt the message illegally. The following algorithm is the GGH cryptosystem [Gil15].

Algorithm 4

1. *Alice generates a good lattice basis B with the shortest vector length r. This basis is going to be the private key.*
2. *Alice finds a unimodular matrix U and calculates the bad basis $B' = U \cdot B$. A check that $\mathscr{H}(B') \approx 0$ may be necessary. The public key is the bad basis with the value of half the length of the shortest vector in the good basis $(B', r/2)$.*
3. *A cleartext message $\mathbf{m}$ from Bob is the coefficients of a lattice vector $\mathbf{p} = \mathbf{m} \cdot B'$.*
4. *After calculating the vector $\mathbf{p}$, the encryption is done by adding a random error vector $\mathbf{e}$ with $\|\mathbf{e}\| < r/2$ to obtain*

$$\mathbf{c} = \mathbf{p} + \mathbf{e}.$$

5. *Alice can decrypt the message by applying the rounding method as follows.*

   a) *Alice solves for $\mathbf{x} = (x_1, \cdots, x_n)$ the equation $\mathbf{x} \cdot B = \mathbf{c}$.*

b) Alice rounds the coefficients vector $\mathbf{x}' = (\lfloor x_1 \rceil, \cdots, \lfloor x_n \rceil)$ *to obtain*

$$\mathbf{p}' = \mathbf{x}' \cdot B.$$

c) Alice can now decrypt the message by solving for $\mathbf{m}'$ *the equation* $\mathbf{p}' = \mathbf{m}' \cdot B'$.

d) According to Theorem 1.25 we have $\mathbf{p} = \mathbf{p}'$. *Thus,* $\mathbf{m} = \mathbf{m}'$.

As an example in Sagemath we consider the basis B and the unimodular matrix U and bad basis B' as in Example 1.19 as follows

```
B = matrix([[-30,0,20], [20,10,30],[20,-130,30]])
U = matrix([[10,39,131],[10,40,133],[3,12,40]])
BB = U*B
```

Now we want to know the length of the shortest vector in B. Thus we define the following routine

```
def Shortest_Length_inMatrix (M):
    l = EuclideanLength(M[0])
    for i in M:
        if (EuclideanLength(i) < l):
            l = EuclideanLength(i)
    return(l)
```

The length of the error vector is not supposed to exceed half of the shortest vector in the lattice. Thus we take the following routine to produce an error vector of integers with a length limit and range of integers.

```
def errorV (dim,length,varRange):
    E = []
    while True:
        for i in range(dim):
            E.append(randint(-varRange,varRange))
        if EuclideanLength(E)<length:
            break
        else:
            E = []
    return(vector(E))
```

Now we define the Encryption function

```
def Encrypt (M,Message,SE):
    PlainText = Message * M
    #print(PlainText)
    Error = errorV(len(Message), SE, 100)
    #print(Error)
    return (PlainText + Error)
```

Assume the message is $(6,7,8)$. Now we try to encrypt it

```
Ciphertext = Encrypt(BB,vector([7,8,9]),Shortest_Length_inMatrix(B)/2)
print(Ciphertext)
    (55516, -297322, 94796)
```

Before defining a decryption function, we need a routine that rounds the entries of a vector to the closest integer

```
def round_vector_to_integers (V):
    L = []
    for i in range(len(V)):
        L.append(floor(V[i]+(1/2)))
    return(vector(L))
```

Now we can define a decryption function

```
def Decrypt (B,BB,Cipher):
    X = B.solve_left(Cipher)
    X = round_vector_to_integers(X)
    Cleartext = X*B
    #print(Cleartext)
    return(BB.solve_left(Cleartext))
```

Now we can decrypt our ciphertext

```
DMessage = Decrypt(B,BB,Ciphertext);DMessage
    (7,8,9)
```

As mentioned before, GGH does not have a security proof. Thus, we will not disscuss the cryptoanalysis of GGH.

2.2 NTRU

NTRU Stands for *Nth Truncated polynomial Ring Units* [SM07]. It was designed by Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman [SM07]. Its security is based on the difficulty of factoring problem in certain polynomial rings [SM07] (Also called NTRU recovery problem [Jef14]). Furthermore, its current security proof is related to SVP and CVP as we will discuss later in this section [Jef14].

2.2.1 Convolutional Polynomial Rings

Before describing the cryptosystems or the related security proof, it is better to describe the polynomial quotient rings that are used by NTRU.

Definition 2.1 (cf. [Jef14]) Let N and q be prime numbers. The ring of *convolutional polynomials of rank* N is the quotient ring

$$R = \frac{\mathbb{Z}[x]}{(x^N - 1)}.$$

The ring of *convolutional polynomials of rank* N *modulo* q is the quotient ring

$$R_q = \frac{\mathbb{Z}_q[x]}{(x^N - 1)}$$

We will denote a polynomial in R or in R_q as a bold italic small letter as

$$\boldsymbol{a}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1}.$$

Note that when we calculate mod $x^N - 1$ we require that

$$\begin{aligned} & x^N - 1 \equiv 0 \pmod{(x^N - 1)} \\ \Longrightarrow \quad & x^N \equiv 1 \pmod{(x^N - 1)}. \end{aligned}$$

Thus, when we have a term x^k, we write

$$k = iN + j \quad \text{where} \quad 0 \leq j < N, \quad \text{and} \quad i, j \in \mathbb{Z}.$$

Then we set

$$x^k = x^{iN+j} = (x^N)^i x^j \equiv 1 \cdot x^j = x^j \pmod{(x^N - 1)}$$ [Jef14].

Addition in R and in R_q is clear. We will denote the multiplication in these rings with $\star$ to distinguish it from the normal polynomial multiplication. We will prove that for

$\boldsymbol{a}(x), \boldsymbol{b}(x) \in R$ we have

$$\boldsymbol{a}(x) \star \boldsymbol{b}(x) = \sum_{k=0}^{N-1} \left(\sum_{i+j \equiv k \pmod N} a_i b_j \right) x^k.$$

Precisely [Jef14],

$$\begin{aligned}
\boldsymbol{a}(x) \star \boldsymbol{b}(x) &= \left(\sum_{i=0}^{N-1} a_i x^i \right) \star \left(\sum_{j=0}^{N-1} b_j x^j \right) \\
&= \sum_{k=0}^{2N-2} \left(\sum_{i+j=k} a_i b_j \right) x^k \\
&= \sum_{k=0}^{N-1} \left(\sum_{i+j=k} a_i b_j \right) x^k + \sum_{k=N}^{2N-2} \left(\sum_{i+j=k} a_i b_j \right) x^{k-N} \\
&= \sum_{k=0}^{N-1} \left(\sum_{i+j=k} a_i b_j \right) + \sum_{k=0}^{N-2} \left(a_i b_j \right) x^k \\
&= \sum_{k=0}^{N-1} \left(\sum_{i+j \equiv k \pmod N} a_i b_j \right) x^k.
\end{aligned}$$

If the $\star$ operation is taken in R_q, we take the ocefficients modulu q at the end as a final step.

For example, let $N = 7$ and

$$\begin{aligned}
\boldsymbol{a}(x) &= x^6 + 3x^5 - 13x^3 + 2x^2 - x + 3 \\
\boldsymbol{b}(x) &= 2x^6 - 2x^4 + 4x^3 + x^2 - 1.
\end{aligned}$$

Then,

$$\boldsymbol{a}(x) \star \boldsymbol{b}(x) = -51x^6 - 4x^5 - 2x^4 + 22x^3 - 27x^2 + 18x + 24$$

in R and

$$\boldsymbol{a}(x) \star \boldsymbol{b}(x) = 4x^6 + x^5 + 3x^4 + 2x^3 + 3x^2 + 3x + 4$$

in R_5.

To implement NTRU, we will need to map elements from R to R_q and in the other direction. We can map from R to R_q by applying $\mod q$ homomorphism. For $\boldsymbol{a}(x), \boldsymbol{b}(x) \in R$, we obtain

$$\begin{aligned}
R &\longrightarrow R_q \\
\boldsymbol{a}(x) &\longrightarrow \boldsymbol{a}_q(x) = \boldsymbol{a}(x) \pmod q \text{ [Jef14].}
\end{aligned}$$

The multiplication and addition can be transferred from R to R_q as follows

$$\begin{aligned}(\boldsymbol{a}(x)+\boldsymbol{b}(x)) \mod q &= (\boldsymbol{a}(x) \mod q) +_q (\boldsymbol{b}(x) \mod q)\\ (\boldsymbol{a}(x)\star\boldsymbol{b}(x)) \mod q &= (\boldsymbol{a}(x) \mod q) \star_q (\boldsymbol{b}(x) \mod q).\end{aligned}$$

Where $+_q$ and $\star_q$ denote the operations in R_q [Jef14].

On the other hand, there are several ways to map elements from R_q to R. We will choose the so-called center-lift for implementation purpose [Jef14].

Definition 2.2 (cf. [Jef14]) Let $\boldsymbol{a}(x)$ be a polynomial in R_q. The *center-lift* of $\boldsymbol{a}(x)$ to R is the unique polynomial $\boldsymbol{a}'(x) \in R$ such that

$$\boldsymbol{a}'(x) \mod q = \boldsymbol{a}(x)$$

in which the coefficients are chosen in the interval

$$-\frac{q}{2} < a_i \leq \frac{q}{2}.$$

Note that the center-lift map is not a homomorphism. It does not preserve the sum and the product. Precisely, the sum or the product of the lifts of two polynomials does not always equal the lift of the sum or the product.

Example 2.3 Let $N = q = 7$ and

$$\boldsymbol{a}(x) = 5x^6+3x^5+5x^4+x^3+x^2+4x+3 \in R_7$$

Then we obtain

$$\text{center-lift of } \boldsymbol{a}(x) = -2x^6+3x^5-2x^4+x^3+x^2-3x+3 \in R$$

In the same manner, let

$$\boldsymbol{b}(x) = 2x^6+4x^5+2x^2+3x+1 \in R_7$$

Thus

$$\text{center-lift of } \boldsymbol{b}(x) = 2x^6-3x^5+2x^2+3x+1 \in R$$

Then

$$\text{center-lift of } (\boldsymbol{a}\star\boldsymbol{b})(x) = -3x^6+x^4-x^3-x^2+x+1 \tag{2.1}$$

However,

$$(\text{center-lift of } \boldsymbol{a})\star(\text{center-lift of } \boldsymbol{b})(x) = 18x^6-14x^5+15x^4-15x^3+6x^2+x-6 \tag{2.2}$$

Note that the polynomials (2.1,2.2) are not equal in R, however, they are congruent in R_q.

2.2.2 GCD & Units in Polynomial Rings

For a prime q, we will describe the devision of two polynomials in the polynomial ring $\mathbb{Z}_q[x]$. Furthermore, We will define the greatest common divisors of two polynomials, which will lead to invertible elements in R_q. Depending on the Euclidean algorithm in polynomials, we will find these inverses.

Theorem 2.4 *[cf. [Jud18]] Let $\boldsymbol{a}(x)$ and $\boldsymbol{b}(x) \neq 0$ be polynomials in $\mathbb{Z}_q[x]$. Then there exist unique polynomials $\boldsymbol{h}(x)$ and $\boldsymbol{r}(x)$ in $\mathbb{Z}_q[x]$ such that*

$$\boldsymbol{a}(x) = \boldsymbol{b}(x) \cdot \boldsymbol{h}(x) + \boldsymbol{r}(x),$$

where either

$$\deg \boldsymbol{r}(x) < \deg \boldsymbol{b}(x),$$

or $\boldsymbol{r}(x) = 0$.

Proof: First, we prove the existence of $\boldsymbol{h}(x)$ and $\boldsymbol{r}(x)$.

If $\boldsymbol{a}(x) = 0$ then

$$\boldsymbol{h}(x) = \boldsymbol{r}(x) = 0.$$

Now we consider $\boldsymbol{a}(x) \neq 0$. Let

$$\deg \boldsymbol{a}(x) = n$$

and

$$\deg \boldsymbol{b}(x) = m.$$

If $m > n$ we take $\boldsymbol{h}(x) = 0$ and $\boldsymbol{r}(x) = \boldsymbol{a}(x)$.

In case $m \leq n$, we will proceed as follows.

1. Let $\boldsymbol{c}_1(x) = (a_n b_m^{-1}) x^{n-m}$.
2. Let
$$\boldsymbol{a}'(x) = \boldsymbol{a}(x) - \boldsymbol{c}_1(x)\boldsymbol{b}(x)$$
where
$$a_i' = \begin{cases} 0 & \text{for } i = n \\ a_i - (a_n b_m^{-1}) & \text{for } n-m \leq i < n \\ a_i & \text{for } i < n-m. \end{cases}$$
3. If $\deg \boldsymbol{a}'(x) < m$, we stop. Otherwise, we repeat the steps (1,2) on $\boldsymbol{a}'(x)$. When

we obtain $\boldsymbol{a}^{(k)}(x)$ with degree $m^{(k)} < n$, we will have also the polynomials

$$\boldsymbol{c}_1(x), \boldsymbol{c}_2(x), \cdots, \boldsymbol{c}_k(x).$$

Now we set

$$\boldsymbol{h}(x) = \sum_{i=1}^{k} \boldsymbol{c}_i(x) \quad \text{and} \quad \boldsymbol{r}(x) = \boldsymbol{a}^{(k)}(x).$$

Now we prove the uniqueness of this division [Jud18]. Assume that we have

$$\boldsymbol{h}'(x), \boldsymbol{r}'(x) \in \mathbb{Z}_q[x]$$

such that

$$\boldsymbol{a}(x) = \boldsymbol{h}'(x)\boldsymbol{b}(x) + \boldsymbol{r}'(x).$$

Then we obtain

$$\boldsymbol{b}(x)\boldsymbol{h}(x) + \boldsymbol{r}(x) = \boldsymbol{b}(x)\boldsymbol{h}'(x) + \boldsymbol{r}'(x).$$

Thus,

$$\boldsymbol{b}(x)[\boldsymbol{h}(x) - \boldsymbol{h}'(x)] = \boldsymbol{r}'(x) - \boldsymbol{r}(x).$$

Consequently,

$$\deg(\boldsymbol{b}(x)[\boldsymbol{h}(x) - \boldsymbol{h}'(x)]) = \deg(\boldsymbol{r}'(x) - \boldsymbol{r}(x)).$$

Clearly

$$\deg(\boldsymbol{b}(x)) \leq \deg(\boldsymbol{b}(x)[\boldsymbol{h}(x) - \boldsymbol{h}'(x)])$$

However, the degrees of $\boldsymbol{r}(x)$ and $\boldsymbol{r}'(x)$ are strictly less than the degree of $\boldsymbol{b}(x)$. Thus, $\boldsymbol{r}(x) = \boldsymbol{r}'(x)$ and $\boldsymbol{h}(x) = \boldsymbol{h}'(x)$. □

The proof of existence in this theorem does not differ too much from the division of polynomials with which most mathematics students are familiar. However, here we take the multiplicative inverse of the leading coefficient in the polynomial $\boldsymbol{b}(x)$. We call $\boldsymbol{h}(x)$ the *quotient* and $\boldsymbol{r}(x)$ the *reminder*.

Example 2.5 Let

$$\boldsymbol{a}(x) = x^5 + 2x^4 + 2x^3 + 5$$

and

$$\boldsymbol{b}(x) = 3x^3 + 5x^2 + 6$$

in $\mathbb{Z}_7[x]$. Then

$$
\begin{array}{r}
5x^2+4x+1 \\
3x^3+5x^2+6\overline{)\,x^5+2x^4+2x^3+5} \\
\underline{-\ (x^5+4x^4+2x^2)\qquad\quad} \\
5x^4+2x^3+5x^2+5 \\
\underline{-\quad (5x^4+6x^3+3x)\quad} \\
3x^3+5x^2+4x+5 \\
\underline{-\quad (3x^3+5x^2+6)} \\
4x+6
\end{array}
$$

Thus,

$$\boldsymbol{a}(x) = (5x^2+4x+1)\boldsymbol{b}(x) + (4x+6).$$

Following Theorem 2.4, we will have the following definitions [Jud18]. A *monic polynomial* $\boldsymbol{b}(x)$ is a *divisor* of $\boldsymbol{a}(x)$ if there exists a polynomial $\boldsymbol{h}(x)$ such that $\boldsymbol{a}(x) = \boldsymbol{b}(x) \cdot \boldsymbol{h}(x)$. We will denote the divisor as $\boldsymbol{b}(x)|\boldsymbol{a}(x)$. A polynomial $\boldsymbol{d}(x)$ is a *common divisor* of polynomials $\boldsymbol{b}(x)$ and $\boldsymbol{a}(x)$ if $\boldsymbol{d}(x)|\boldsymbol{a}(x)$ and $\boldsymbol{d}(x)|\boldsymbol{b}(x)$. A monic polynomial $\boldsymbol{d}(x)$ is the *greatest common divisor* of two polynomials if any other common divisor $\boldsymbol{d'}(x)$ also divides $\boldsymbol{d}(x)$. Two polynomials $\boldsymbol{a}(x)$ and $\boldsymbol{b}(x)$ are *comprime* if $\gcd(\boldsymbol{a}(x), \boldsymbol{b}(x)) = 1$.

Euclidean and extended Euclidean Algorithms are understood in Integers [McA11]. In addition, Euclidean algorithm can be implemented to find the gcd of two polynomials. Furthermore, the extended Euclidean algorithm can be implemented to find for given polynomials $\boldsymbol{a}(x)$ and $\boldsymbol{b}(x)$ another two polynomials $\boldsymbol{s}(x)$ and $\boldsymbol{t}(x)$ such that

$$\boldsymbol{a}(x) \cdot \boldsymbol{s}(x) + \boldsymbol{b}(x) \cdot \boldsymbol{t}(x) = \gcd(\boldsymbol{a}(x), \boldsymbol{b}(x)) \text{ [Jef14].} \tag{2.3}$$

Example 2.6 Consider the polynomials

$$\boldsymbol{a}(x) = x^7 + 9x^6 + x^5 + 6x^4 + 4x^2 + 7$$

and

$$\boldsymbol{b}(x) = 10x^4 + 6x^3 + 6x + 1$$

in $\mathbb{Z}_{11}[x]$. For the algorithm, we form a table with five columns as in Table 2.1. The first column is an enumerator, the second column is for quotients, the third is for reminders, and the fourth and fifth are for the final results. The first two rows are filled as an initial state. For $i > 2$, we fill the fields as follows

$$\begin{aligned}
\boldsymbol{q}_i &= \text{ quotient of the division } \boldsymbol{r}_{i-2}/\boldsymbol{r}_{i-1} \\
\boldsymbol{r}_i &= \boldsymbol{r}_{i-2} - (\boldsymbol{q}_i \cdot \boldsymbol{r}_{i-1}) \\
\boldsymbol{u}_i &= \boldsymbol{u}_{i-2} - (\boldsymbol{q}_i \cdot \boldsymbol{u}_{i-1}) \\
\boldsymbol{v}_i &= \boldsymbol{v}_{i-2} - (\boldsymbol{q}_i \cdot \boldsymbol{v}_{i-1}).
\end{aligned}$$

We stop for $\boldsymbol{r}_k(x) = 0$ for some k to obtain

$$\boldsymbol{r}_{k-1}(x) = r_0 + r_1 x + \cdots + r_m x^m.$$

Thus,

$$\gcd(\boldsymbol{a}(x), \boldsymbol{b}(x)) = \boldsymbol{r}_{k-1}(x)/r_m$$

and

$$\boldsymbol{s}(x) = \boldsymbol{u}_{k-1}(x)/r_m \quad \text{and} \quad \boldsymbol{t}(x) = \boldsymbol{v}_{k-1}(x)/r_m.$$

In our example and from Table 2.1 we obtain

i	$\boldsymbol{q}$	$\boldsymbol{r}$	$\boldsymbol{u}$	$\boldsymbol{v}$
1		$x^7+9x^6+x^5+6x^4+4x^2+7$	1	0
2		$10x^4+6x^3+6x+1$	0	1
3	$10x^3+7x^2+8x+3$	$7x^3+4x^2+7x+4$	1	x^3+4x^2+3x+8
4	$3x+7$	$6x^2+6$	$8x+4$	$8x^4+3x^3+7x^2+10x$
5	$3x+8$	0		

Table 2.1: Extended Euclidean Algorithm for two polynomials in $F_{11}[x]$.

$$\gcd(\boldsymbol{a}(x), \boldsymbol{b}(x)) = x^2+1 = \boldsymbol{a}(x)\cdot(5x+8) + \boldsymbol{b}(x)\cdot(5x^4+6x^3+3x^2+9x)$$

Theorem 2.7 (cf. [Jef14]) *Let q be a prime number. Then $\boldsymbol{a}(x) \in R_q$ is a unit (has a multiplicative inverse) if and only if*

$$\gcd(\boldsymbol{a}(x), (x^N-1)) = 1 \tag{2.4}$$

in $\mathbb{Z}_q[x]$.

Proof: If $\boldsymbol{a}(x)$ is a unit in R_q, then there exists an element $\boldsymbol{a}^{-1}(x) \in R_q$ such that

$$\boldsymbol{a}(x) \star \boldsymbol{a}^{-1}(x) \equiv 1 \qquad \in R_q \tag{2.5}$$

$$\implies \quad \boldsymbol{a}(x)\cdot\boldsymbol{a}^{-1}(x) + \boldsymbol{k}(x)(x^N-1) = 1 \quad \in \mathbb{Z}_q[x] \tag{2.6}$$

where $\boldsymbol{k}(x)$ is in $\mathbb{Z}_q[x]$. Thus, $\gcd(\boldsymbol{a}(x), (x^N-1)) = 1$ because otherwise we will be able to evenly divide both sides of the equation (2.6) by an element in $\mathbb{Z}_q$.
Now, assume that

$$\gcd(\boldsymbol{a}(x), (x^N-1)) = 1$$

in $\mathbb{Z}_q[x]$. According to Euclidean algorithm, there exist $\boldsymbol{s}(x)$ and $\boldsymbol{t}(x)$ elements in $\mathbb{Z}_q[x]$ such that

$$\boldsymbol{a}(x)\cdot\boldsymbol{s}(x) + (x^N-1)\boldsymbol{t}(x) = 1 \tag{2.7}$$

When we take the equation (2.7) modulus (x^N-1), we will conclude that

$$\boldsymbol{a}(x)\star\boldsymbol{s}(x)\equiv 1\in R_q.$$

Thus,

$$\boldsymbol{a}^{-1}(x)=\boldsymbol{s}(x)$$

is the multiplicative inverse of $\boldsymbol{a}(x)$ in R_q. □

The previous theorem gives us a methodology for inverting elements in R_q. First, we check invertibility of a polynomial $\boldsymbol{a}(x)\in R_q$ by simply lift this polynomial to $\mathbb{Z}_q[x]$ and then check that the $\gcd(\boldsymbol{a}(x),(x^N-1))$ equals 1. Furthermore, we can calculate the multiplicative inverse of $\boldsymbol{a}(x)$ by applying the extended Euclidean algorithm as in Example 2.6 [Jef14].

2.2.3 NTRU Cryptosystem

To apply NTRU we need four given positive integer parameters (N,p,q,d) such that N,p and q are primes. Furthermore, we require that $q>(6d+1)p$. We define the convolutional polynomial rings

$$R=\frac{\mathbb{Z}[x]}{(x^n-1)},\qquad R_p=\frac{\mathbb{Z}_p[x]}{(x^n-1)},\qquad R_q=\frac{\mathbb{Z}_q[x]}{(x^n-1)}\text{ [Jef14].}$$

To lift a polynomial $\boldsymbol{a}(x)\in R$ to R_p or R_q we simply reduce the coefficients modulo p or q and we will denote these polynomials by $\boldsymbol{a}_p(x),\boldsymbol{a}_q(x)$ respectively [Jef14]. We use center-lifts to represent $\boldsymbol{a}_p(x)$ or $\boldsymbol{a}_q(x)$ in R [Jef14].

Definition 2.8 (cf. [Jef14]) For given positive integers d_1 and d_2, we define *ternary polynomials* as follows

$$\mathscr{T}(d_1,d_2)=\left\{\boldsymbol{a}(x)\in R:\begin{array}{l}\boldsymbol{a}(x)\text{ has }d_1\text{ coefficients equal to }+1,\\ \boldsymbol{a}(x)\text{ has }d_2\text{ coefficients equal to }-1,\\ \boldsymbol{a}(x)\text{ has all other coefficients equal to }0.\end{array}\right\}$$

Now we can introduce the NTRU cryptosystem according to [Jef14].

Algorithm 5

1. *Receiver chooses randomly $\boldsymbol{f}(x)\in\mathscr{T}(d+1,d)$ such that $\boldsymbol{f}_p(x)$ and $\boldsymbol{f}_q(x)$ are invertible in R_p and R_q with inverses $\boldsymbol{f}_p^{-1}(x)\in R_p$ and $\boldsymbol{f}_q^{-1}(x)\in R_q$. Furthermore, receiver chooses $\boldsymbol{g}(x)\in\mathscr{T}(d,d)$ randomly.*

2. *The public key is*
$$\boldsymbol{h}(x) = \boldsymbol{f}_q^{-1}(x) \star \boldsymbol{g}(x) \in R_q.$$
3. *The private key is* $(\boldsymbol{f}(x), \boldsymbol{f}_p^{-1}(x))$.
4. *A message is a polynomial* $\boldsymbol{m}(x) \in R$ *with coefficients*
$$-p/2 < m_i \leq p/2.$$
In other words, a message is a polynomial in R *that is the center-lift of a polynomial in* R_p.
5. *to encrypt* $\boldsymbol{m}(x)$*, sender chooses a random polynomial* $\boldsymbol{r}(x)$*. Then the ciphertext is*
$$\boldsymbol{c}(x) \equiv p\boldsymbol{h}(x) \star \boldsymbol{r}(x) + \boldsymbol{m}(x) \pmod{q} \in R_q.$$
6. *Decrypting the ciphertext goes as follows. Receiver first computes*
$$\boldsymbol{l}(x) \equiv \boldsymbol{f}(x) \star \boldsymbol{c}(x) \pmod{q} \in R_q.$$
7. *Second step in decryption is to center lift* $\boldsymbol{l}(x)$ *to a polynomial in* R.
8. *Finally, the receiver computes*
$$\boldsymbol{m}'(x) \equiv \boldsymbol{f}_p^{-1}(x) \star \boldsymbol{l}(x) \pmod{p} \in R_p$$
9. *If parameters are chosen properly, then* $\boldsymbol{m}(x) = \boldsymbol{m}'(x)$.

To see why this system works, we will describe the calculation of $\boldsymbol{m}'(x)$ as follows [Jef14].

$$\begin{aligned}
\boldsymbol{a}(x) &\equiv \boldsymbol{f}(x) \star \boldsymbol{c}(x) \pmod{q} \\
&\equiv \boldsymbol{f}(x)\,(p\boldsymbol{h}(x) \star \boldsymbol{r}(x) + \boldsymbol{m}(x)) \pmod{q} \\
&\equiv p\boldsymbol{f}(x) \star \boldsymbol{f}_q^{-1}(x) \star \boldsymbol{g}(x) \star \boldsymbol{r}(x) + \boldsymbol{f}(x) \star \boldsymbol{m}(x) \pmod{q} \\
&\equiv p\boldsymbol{g}(x) \star \boldsymbol{r}(x) + \boldsymbol{f}(x) \star \boldsymbol{m}(x) \pmod{q}.
\end{aligned}$$

Let's assume that
$$\boldsymbol{a}(x) = p\boldsymbol{g}(x) \star \boldsymbol{r}(x) + \boldsymbol{f}(x) \star \boldsymbol{m}(x)$$
is calculated in R.

We will check the magnitude (the absolute value) of the greatest coefficient in $\boldsymbol{a}(x)$. Since $\boldsymbol{g}(x), \boldsymbol{r}(x) \in \mathscr{T}(d,d)$ and if all the coefficients in each of these two polynomials gather in the same term x^k in the product, this coefficient will equal to $2d$. For the other term in $\boldsymbol{a}(x)$, the $\boldsymbol{f}(x)$ is in $\mathscr{T}(d+1,d)$ and the coefficients of $\boldsymbol{m}(x)$ are between $-\frac{1}{2}p$ and $\frac{1}{2}p$. Thus the largest possible coefficient of a term $x^{k'}$ in $\boldsymbol{f}(x) \star \boldsymbol{m}(x)$ is $(2d+1)\frac{1}{2}p$. Assuming that $k = k'$, the largest possible magnitude of a term in $\boldsymbol{a}(x)$ calculated in R

is

$$2dp + (2d+1)\frac{1}{2}p = \left(3d + \frac{1}{2}\right)p.$$

Thus, by choosing $q > (6d+1)p$ we are ensuring that any coefficient in $\boldsymbol{a}(x)$ calculated in R is less than $\frac{1}{2}q$. Consequently, calculating $\boldsymbol{a}(x)$ in R_q and then center-lifting it to R will give us the same coefficients [Jef14].

Now we check the the following product in R_p

$$\begin{aligned}
\boldsymbol{m}'(x) &\equiv \boldsymbol{f}_p^{-1}(x) \star \boldsymbol{a}(x) \pmod{p} \\
&\equiv \boldsymbol{f}_p^{-1}(x)\,(p\boldsymbol{g}(x) \star \boldsymbol{r}(x) + \boldsymbol{f}(x) \star \boldsymbol{m}(x)) \pmod{p} \\
&\equiv \boldsymbol{f}_p^{-1}(x) \star \boldsymbol{f}(x) \star \boldsymbol{m}(x) \pmod{p} \\
&\equiv \boldsymbol{m}(x) \pmod{p} \text{ [Jef14].}
\end{aligned}$$

Note that the polynomials that will be used in NTRU are all *ternary*. When we move these polynomials to R_q we simply take the coefficients $\bmod q$. Thus, the non-zero coefficients of these polynomials are 1 or $q-1$ when converted to R_q. To preserve these properties, when we move from R_q to R we first center-lift the coefficients. Consequently, if we consider q to be a composite number, then the non-zero coefficients in R_q are Units in $\mathbb{Z}/q\mathbb{Z}$. Because 1 is a unit and

$$(q-1)^2 = q^2 - 2q + 1 \equiv 1 \pmod{q}.$$

This applies also to R_p. Consequently, this will allow us to apply Theorem 2.4 and search for the greatest common divisors even though q is a composite number. Thus, we can implement NTRU with composite q and p. And define the quotient rings with

$$R_q = \frac{\mathbb{Z}/q\mathbb{Z}[x]}{x^N - 1} \qquad R_p = \frac{\mathbb{Z}/p\mathbb{Z}[x]}{x^N - 1}$$

with further condition $\gcd(N,q) = \gcd(q,p) = 1$. This is the general choice of of NTRU parameters and rings in [Jef14].

2.2.4 Software Implementation

Rather than writing an NTRU example by hand, we will have an attempt by defining some routines in `SageMath` to generate NTRU keys with adjustable initial parameters (N,p,q,d). Without loss of generality, the parameters p and q are supposed to be primes. For flexible conversion between rings, we will define the polynomial rings instead of the quotient polynomial rings. The $\pmod{(x^n-1)}$ calculations are taken in each step and in several rings. Thus, we will consider the polynomial rings in these routines as quotient polynomial rings. Further, in the construction of the rings, we will

consider the variables `x`, `px`, and `qx` for the rings R, R_p, and R_q respectively. Further, we will need the package `random`.

```
import random
N = 23
p = 3
q = 257
d = 5
R.<x>      = PolynomialRing(ZZ)
RP.<px>    = PolynomialRing(Zmod(p))
RQ.<qx>    = PolynomialRing(Zmod(q))
Modulus = x^N -1
Modulus_p = RP(Modulus)
Modulus_q = RQ(Modulus)
```

The following routine returns random ternary polynomials in ring `R` where the number of $+1$'s is `a` and the number of -1's is `b`. Thus, we will consider the output of this routine to be a random element in $\mathscr{T}(a,b)$.

```
def aOnes_bMinuses_Polynomial (N, a, b, R):
    ones = []
    for i in range(a):
        ones.append(1)
    minuses = []
    for i in range(b):
        minuses.append(-1)
    zeroes = []
    for i in range(N-1-a-b):
        zeroes.append(0)
    coefficients = ones+minuses+zeroes
    random.shuffle(coefficients)
    Poly = 0
    for i in range(len(coefficients)):
        Poly += coefficients[i] * var(R.variable_name())^i
    return(R(Poly))
```

Now we need a routine to create NTRU key building blocks. Precisely, we need a polynomial in $\mathscr{T}(d+1,d)$ which is invertible in R_p and R_q. To do so, the following routine applies rejection sampling [Inf23]. In other words, we create randomly many polynomials in $\mathscr{T}(d+1,d)$ and check the invertibility in R_p and in R_q until we reach a polynomial that satisfies the requirements. The output of this routine is a dictionary with a polynomial and its inverses in R_p and R_q. In addition, we will obtain a random polynomial in $\mathscr{T}(d,d)$.

```
def NTRU_Key_Gen (N,p,q,d):
    F = 0
    Check = False
    while (not Check):
        F = aOnes_bMinuses_Polynomial(N,d,d-1,R)
        if ( (gcd(F,Modulus) ==1)       and
             (gcd(Modulus_p,RP(F)) ==1) and
             (gcd(Modulus_q,RQ(F)) ==1)):
            Check = True
    g = aOnes_bMinuses_Polynomial(N,d,d,R)
    return({"f" : F,
            "f_p" : RP(F),
            "inv_f_p" : inverse_mod(RP(F),Modulus_p),
            "f_q":RQ(F),
            "inv_f_q" : inverse_mod(RQ(F),Modulus_q),
            "g" : g})
```

The input of the following routine is a dictionary of key material while the output is the corresponding public key.

```
def NTRU_public_key (Key):
    return(RQ(Key["g"]) * Key["inv_f_q"] % Modulus_q)
```

The following is the NTRU encryption function, considering a public key and a message with predefined parameters. The output is the ciphertext which is a polynomial in R_q.

```
def NTRU_Encrypt(N,p,q,d,Public_Key,Message):
    r = aOnes_bMinuses_Polynomial(N,d,d,R)
    c = p * R(Public_Key) * r + Message
    c = c % R(x^N -1)
    return(RQ(c) % Modulus_q)
```

Further, we will define a decryption function. Here we center-lift the polynomials to move them from R_p or R_q to R.

```
def NTRU_Decrypt (N,p,q,d,Key,Cipher):
    a = (RQ(Key["f"]) * Cipher) % Modulus_q
    a = a.map_coefficients(lambda c: c.lift_centered(), ZZ)
    a = R(a)
    b = R(Key["inv_f_p"]) * a
    MESSAGE = RP(b) % Modulus_p
```

```
    MESSAGE = MESSAGE.map_coefficients(lambda c: c.lift_centered(), ZZ)
    return(R(MESSAGE))
```

The recommendations of the NTRU encryption scheme will lead to huge polynomials [HSW05]. Since our work is only to demonstrate the functionality of NTRU, we will consider smaller parameters. For instance, we will assume $N = 23, p = 3, q = 257$, and $d = 5$. Now we will create one key using the previously defined parameters.

```
NTRU_Key = NTRU_Key_Gen(N,p,q,d)
print(NTRU_Key["f"])
```

Each time we run these routines we obtain new NTRU key material. The current given polynomial is

$$\boldsymbol{f}(x) = x^{20} - x^{17} + x^{16} + x^{13} + x^{11} - x^{10} - x^9 - x^7 - x^5 + x^3 + x.$$

With inverses

$$\begin{aligned}\boldsymbol{f}_3^{-1} = {} & 2x^{22} + 2x^{21} + x^{20} + x^{19} + 2x^{18} + 2x^{17} + 2x^{16} + 2x^{15} + x^{13} + x^{12} + x^{11} \\ & + 2x^8 + x^7 + 2x^6 + x^4 + 2 \quad \in R_3\end{aligned}$$

and

$$\begin{aligned}\boldsymbol{f}_{257}^{-1} = {} & 40x^{22} + 6x^{21} + 120x^{20} + 227x^{19} + 40x^{18} + 164x^{17} + 29x^{16} + 187x^{15} + 41x^{14} \\ & + 158x^{13} + 95x^{12} + 34x^{11} + 224x^{10} + 41x^9 + 204x^8 + 53x^7 + 158x^6 + 84x^5 + 238x^4 \\ & + 152x^3 + 222x^2 + 22x + 32 \quad \in R_{257}\end{aligned}$$

We may check the following multiplications to ensure the required properties.

```
NTRU_Key["f_p"] * NTRU_Key["inv_f_p"] % Modulus_p
    1
NTRU_Key["f_q"] * NTRU_Key["inv_f_q"] % Modulus_q
    1
```

Now we create the public key from the previously defined key material

```
    NTRU_Public = NTRU_public_key(N,p,q,d,NTRU_Key)
    print(NTRU_Public)
```

to obtain the public key

$$\begin{aligned}\boldsymbol{h}(x) = {} & 17x^{22}+241x^{21}+214x^{20}+209x^{19}+51x^{18}+224x^{17}+204x^{16}+81x^{15}+75x^{14}+220x^{13} \\ & +129x^{12}+111x^{11}+77x^{10}+117x^{9}+148x^{8}+251x^{7}+58x^{6}+81x^{5}+76x^{4}+176x^{3} \\ & +256x^{2}+23x+45 \in R_{257}.\end{aligned}$$

To test our functions, we need a message that we will create randomly.

```
Pluses = randint(0,N)
Minuses = randint(0,N-Pluses)
Message = aOnes_bMinuses_Polynomial(N,Pluses,Minuses,R)
print(Message)
```

Let's consider the following polynomial message.

$$\begin{aligned}\boldsymbol{m}(x) = {} & x^{22}+x^{21}+x^{20}-x^{19}+x^{18}-x^{17}-x^{16}+x^{15}-x^{14}+x^{13}-x^{12}+x^{11}+x^{10} \\ & -x^{9}+x^{8}-x^{7}-x^{6}-x^{5}+x^{4}+x^{3}+x^{2}-x+1 \quad \in R\end{aligned}$$

Now we can encrypt this message using the previously calculated public key and the predefined encryption function.

```
CipherText = NTRU_Encrypt(N,p,q,d,NTRU_Public,Message)
print(CipherText)
```

We obtain the following ciphertext

$$\begin{aligned}\boldsymbol{c}(x) = {} & x^{22}+121x^{21}+201x^{20}+54x^{19}+43x^{18}+152x^{17}+200x^{16}+28x^{15}+233x^{14} \\ & +199x^{13}+12x^{11}+71x^{10}+64x^{9}+176x^{8}+6x^{7}+86x^{6}+66x^{5}+138x^{4}+225x^{3} \\ & +103x^{2}+26x+111 \quad \in R_{257}.\end{aligned}$$

Now we will try to decrypt this ciphertext and then compare the output with the original message.

```
    MESSAGE = NTRU_Decrypt(N,p,q,d,NTRU_Key,CipherText)
    MESSAGE == Message
        True
```

2.2.5 NTRU Cryptoanalysis

From the construction of the NTRU public key, we conclude that

$$\boldsymbol{f}(x) \star \boldsymbol{h}(x) \equiv \boldsymbol{g}(x) \pmod{q}. \tag{2.8}$$

Thus, breaking the NTRU cryptosystem can be summarized as follows. For a given $\boldsymbol{h}(x)$ find two ternary polynomials $\boldsymbol{f}(x)$ and $\boldsymbol{g}(x)$ such that $\boldsymbol{f}(x) \star \boldsymbol{h}(x) = \boldsymbol{g}(x)$ where $\boldsymbol{f}(x) \in \mathscr{T}(d+1,d)$ and $\boldsymbol{g} \in \mathscr{T}(d,d)$. This is called the *NTRU recovery problem*. Note that if there exist two polynomials that satisfy the equation (2.8) then

$$x^k \star \boldsymbol{f}(x) \star \boldsymbol{h}(x) \equiv x^k \star \boldsymbol{g}(x) \pmod{q}$$

for any positive integer k.

Definition 2.9 Let k be a positive integer and $\boldsymbol{p}(x)$ be a polynomial in R. The polynomial $x^k \star \boldsymbol{p}(x)$ is called a *rotation* of $\boldsymbol{p}(x)$

Let's assume that the polynomial $\boldsymbol{f}(x)$ which is used in the construction of the NTRU public key $\boldsymbol{h}(x)$ and its rotations $x^k \boldsymbol{f}(x)$ are the only solutions that will produce a ternary polynomial in the right side of the equation (2.8). Later we will prove that it is extremely rare to find other solutions. Thus finding any of these rotations will lead us to the private key polynomials by checking the possible rotations [Jef14]. In the following, we will implement a successful brute force attack and we will discuss the factors to harden this implementation.

Example 2.10 Considering the codes which were used in Section 2.2.4, we will produce new NTRU key material with parameters $(7,3,257,2)$. Let the key material be the polynomials

$$\begin{aligned}
\boldsymbol{f}(x) &= -x^5 - x^4 + x^2 + x + 1 \in R \\
\boldsymbol{h}(x) &= x^6 + x^4 + 255x^3 + x^2 + x + 255 \in R_q \\
\boldsymbol{g}(x) &= x^3 - x^2 - x + 1 \in R.
\end{aligned}$$

We will apply the following routine to produce all possible polynomials in $\mathscr{T}(3,2)$.

```
import itertools
def List_all_T_PrivateKeys(N,p,q,d):
    RandomTernary = aOnes_bMinuses_Polynomial(N,d+1,d,R)
    RandomTernaryList = RandomTernary.list()
    while (len(RandomTernaryList) < N):
```

```
        RandomTernaryList.append(0)
    APCC = list(itertools.permutations(RandomTernaryList))
    All_Ternaries = []
    Poly = 0
    for i in range(len(APCC)):
        Poly = 0
        for j in range(len(APCC[i])):
            Poly += APCC[i][j]*x^j
        All_Ternaries.append(R(Poly))
    return(All_Ternaries)
```

We can repeat the same code by changing `d+1` by `d` in the first line to obtain all the elements in $\mathscr{T}(d,d)$. By doing so, we will have a second routine to list all possible elements in $\mathscr{T}(d,d)$ which we will call `List_all_T_g`. To start the comparison, we need the following lists.

```
fList = List_all_T_PrivateKeys(N,p,q,d)
gList = List_all_T_g(N,p,q,d)
```

We run the following loop to find a possible private key.

```
fPoly=0
gPoly = 0
for i in fList :
    fPoly = i
    PPP = (RQ(i) * NTRU_Public) % Modulus_q
    gPoly = gPoly.map_coefficients(lambda c : c.lift_centered(),ZZ)
    gPoly = R(gPoly)
    if (gPoly in glist):
        break
    gPoly = 0
print(gPoly)
print(fPoly)
```

In the last implementation, the result was as follows

$$\boldsymbol{g}^*(x) = x^5 - x^4 - x^3 + x^2$$
$$\boldsymbol{f}^*(x) = -x^6 - x^5 + x^3 + x^2 + x$$

Note that

$$\boldsymbol{f}(x) = x^6 \star \boldsymbol{f}^*(x), \quad \text{and} \quad \boldsymbol{g}(x) = x^6 \star \boldsymbol{g}^*(x)$$

The previous example is designed to be easily solved. However, to know how hard the problem is, we will check the factors that make it harder. Precisely, there may be better ways to produce elements of $\mathscr{T}(d+1,d)$ and to check that the product $\boldsymbol{f}(x)\star\boldsymbol{h}(x)$ is ternary. However, we must check elements of $\mathscr{T}(d+1,d)$.

Given N, d_1, and d_2, we can form a ternary polynomial in R by choosing d_1 coefficients to give them the value 1 and then choose d_2 coefficients from $N-d_1$ to give them the value -1 while the rest coefficients will take the value 0. Thus

$$\#\mathscr{T}(d_1,d_2)=\binom{N}{d_1,d_2}=\frac{N!}{d_1!d_2!(N-d_1-d_2)!}\text{ [Jef14].}$$

Since all the N rotations of $\boldsymbol{f}(x)$ are successfull choices, an evasedropper must Check $\#\mathscr{T}(d+1,d)/N$ to find a rotation of $\boldsymbol{f}(x)$ [Jef14]. According to the recommendations [HSW05] the minimum chosen parameters are

$$(N,p,q,d)=(251,3,293,8). \tag{2.9}$$

Thus, the number of checks to be done to implement a successful brute-force attack is

$$\frac{\#\mathscr{T}(9,8)}{251}=\frac{\binom{251}{9,8}}{251}\approx 2^{100.8}.$$

The same paper gives a better parameter tuple where

$$(N,p,q,d)=(397,3,659,12) \tag{2.10}$$

which will lead to the following number of checks in a successful brute force attack

$$\frac{\#\mathscr{T}(13,12)}{397}=\frac{\binom{397}{13,12}}{397}\approx 2^{153}.$$

The polynomials $\boldsymbol{f}(x)$ and its rotations are probably the only decryption keys [Jef14]. We will discuss the occurrence of other decryption keys probabilistically. Note that for a polynomial $\boldsymbol{f}(x)\in\mathscr{T}(d+1,d)$ to be a possible decryption key, it must satisfy that the product

$$\boldsymbol{f}(x)\star\boldsymbol{h}(x)\pmod q \text{ is in } \mathscr{T}(d,d) \text{ [Jef14].} \tag{2.11}$$

Let X denote that a randomly chosen polynomial $\boldsymbol{f}(x)$ from $\mathscr{T}(d+1,d)$ is a decryption key. We will first show that

$$P(X)\approx\left(\frac{3}{q}\right)^N.$$

The number of ternary polynomials in R is much greater than the number of elements in

$\mathscr{T}(d,d)$. We will consider that the coefficients of the product (2.11) are independently uniformly distributed modulo q. To ease calculations, we will relax the condition (2.11) to be a ternary polynomial. Then the probability of obtaining a ternary coefficient is $3/q$. Since we have N terms in the product (2.11), the probability that it is ternary is $(3/q)^N$ [Jef14].

However, by definition of probability, we have

$$P(X) = \frac{\text{\#Decryption Keys}}{\#\mathscr{T}(d+1,d)}.$$

Thus, we obtain

$$\begin{aligned}\text{\#Decryption Keys} &\approx P(X) \times \#\mathscr{T}(d+1,d)\\ &= \left(\frac{3}{q}\right)^N \binom{N}{d+1,d} \text{ [Jef14].}\end{aligned}$$

In our parameters (2.9) we obtain

$$\left(\frac{3}{293}\right)^{251} \binom{251}{9,8} \approx 2^{-1558}.$$

In our parameters (2.10) we obtain

$$\left(\frac{3}{659}\right)^{397} \binom{397}{13,12} \approx 2^{-2934}.$$

Thus, the design of the public key depending on the private key $\boldsymbol{f}(x)$ will probably have no other decryption keys but the rotations of $\boldsymbol{f}(x)$.

2.2.6 NTRU Lattice

In this section, we will represent two polynomials $\boldsymbol{a}(x)$ and $\boldsymbol{b}(x)$ from R as vectors of coefficients as follows

$$(\boldsymbol{a},\boldsymbol{b}) = (a_0, a_1, \cdots, a_{N-1}, b_0, b_1, \cdots, b_{N-1}).$$

Assuming that $\boldsymbol{h}(x)$ is an NTRU public key generated by the private key $(\boldsymbol{f}(x), \boldsymbol{g}(x))$, we will show that there is a lattice for which the vector $(\boldsymbol{f},\boldsymbol{g})$ is a relatively short vector (or probably, the shortest vector) [Jef14].

Firstly, we will define the *NTRU Lattice* associated to a public key $\boldsymbol{h}(x)$ with parameters

(N, p, q, d) to be the lattice generated by the $2N \times 2N$ matrix

$$M_{\boldsymbol{h}}^{\text{NTRU}} = \begin{pmatrix} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1} \\ 0 & 1 & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0 \\ 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{pmatrix} \text{ [Jef14].}$$

This matrix is denoted as the NTRU matrix. By denoting the NTRU Lattice by $L_{\boldsymbol{h}}{}^{\text{NTRU}}$ we can write

$$L_{\boldsymbol{h}}{}^{\text{NTRU}} = \mathscr{L}(M_{\boldsymbol{h}}{}^{\text{NTRU}}).$$

Note that the NTRU matrix can be abbreviated as

$$M_{\boldsymbol{h}}{}^{\text{NTRU}} = \begin{pmatrix} I & \boldsymbol{h} \\ 0 & qI \end{pmatrix} \text{ [Jef14].}$$

by decomposing this matrix into four $N \times N$ block matrices as follows

- The top left block is the identity matrix.
- The top right block is the vectors of the cyclic permutations of the coefficients of $\boldsymbol{h}(x)$.
- The lower left block is the zero matrix.
- The lower right block is q times the identity matrix.

Now we will check that there exists a vector $(\boldsymbol{a}, \boldsymbol{b})$ representing two polynomials $\boldsymbol{a}(x)$ and $\boldsymbol{b}(x)$ from R such that

$$(\boldsymbol{a}, \boldsymbol{b}) M_{\boldsymbol{h}}{}^{\text{NTRU}} = (\boldsymbol{f}, \boldsymbol{g}) \text{ [Jef14].}$$

Since

$$\boldsymbol{f}(x) \star \boldsymbol{h}(x) \equiv \boldsymbol{g} \pmod{q}$$

in R_q, then there exists a polynomial $\boldsymbol{u}(x)$ in R such that

$$\boldsymbol{f}(x) \star \boldsymbol{h}(x) = \boldsymbol{g}(x) + q\boldsymbol{u}(x) \tag{2.12}$$

in R. Let's check if the equation

$$(\boldsymbol{f}, -\boldsymbol{u}) M_{\boldsymbol{h}}{}^{\text{NTRU}} = (\boldsymbol{f}, \boldsymbol{g}) \tag{2.13}$$

is fulfilled. The first N coordinates from the left side product in equation (2.13) are the

coefficients of the polynomial $\boldsymbol{f}(x)$. Let

$$i \in \{N+1, N+2, \cdots, 2N\}$$

and

$$k = (i-1) \mod N,$$

the ith coordinate of the product in the left side of the equation (2.13) has the value

$$h_k f_0 + h_{k-1} f_1 + \cdots h_{k+1} f_{N-1} - q u_k$$

which is according to the equation (2.12) the kth coefficient of $\boldsymbol{g}(x)$. Thus, the second N coordinates of the product in the left side of the equation (2.13) are the coefficients of the polynomial $\boldsymbol{g}(x)$.

Now we want to compare the length of the vector $(\boldsymbol{f}, \boldsymbol{g})$ with the Gaussian expected shortest length in equation (1.5). Note that

$$\det(M_{\boldsymbol{h}}^{\text{NTRU}}) = q^N$$

because it is an upper triangular matrix. Assume that $\mathbf{v}$ is the shortest vector in $L_{\boldsymbol{h}}^{\text{NTRU}}$. Thus

$$\|\mathbf{v}\| \approx \sigma(M_{\boldsymbol{h}}^{\text{NTRU}}) = \sqrt{\frac{qN}{\pi e}}. \tag{2.14}$$

On the other hand

$$\|(\boldsymbol{f}, \boldsymbol{g})\| = \sqrt{1+4d}. \tag{2.15}$$

The comparison between these two values is done in [Jef14] by taking d as a function in N with higher values than implementation requirements. However, we will take numerical examples to check this. In our example (2.9), we obtain

$$\|(\boldsymbol{f}, \boldsymbol{g})\| \overset{(2.15)}{=} \sqrt{33} \approx 5.7 < 92.8 \overset{(2.14)}{\approx} \|\mathbf{v}\|.$$

In our example (2.10), we have

$$\|(\boldsymbol{f}, \boldsymbol{g})\| = \sqrt{49} = 7 < 175 \approx \|\mathbf{v}\|.$$

These values show us that the expected length of a shortest vector in $L_{\boldsymbol{h}}^{\text{NTRU}}$ is bigger than the length of the vector $(\boldsymbol{f}, \boldsymbol{g})$ which leads to the result that the vector $(\boldsymbol{f}, \boldsymbol{g})$ and the similar vectors related to the rotations of the polynomial $\boldsymbol{f}(x)$ are extremely short vectors in the lattice $L_{\boldsymbol{h}}^{\text{NTRU}}$ [Jef14]. In addition, these vectors are probably the shortest vectors in this lattice [Jef14].

2.2.7 LLL-Algorithm attack on NTRU

A cryptoanalysis attack against NTRU can be designed by implementing lattice reduction [Jef14]. In the following example, we will crack an NTRU cryptosystem by using the LLL-algorithm to find the extreme short vector $(\boldsymbol{f}, \mathbf{g})$ and its rotations from the NTRU lattice base.

Example 2.11 Let's consider an NTRU construction polynomials with the parameters

$$(N, p, q, d) = (7, 3, 47, 2)$$

as follows

$$\begin{aligned}
\boldsymbol{f}(x) &= -x^4 - x^3 + x^2 + x + 1, \\
\boldsymbol{g}(x) &= -x^5 - x^4 + x^2 + x, \\
\boldsymbol{h}(x) &= 12x^6 + 4x^5 + 23x^4 + 38x^3 + x^2 + 18x + 45.
\end{aligned}$$

The related NTRU lattice is generated by the following Matrix.

$$M = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 45 & 18 & 1 & 38 & 23 & 4 & 12 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 12 & 45 & 18 & 1 & 38 & 23 & 4 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 4 & 12 & 45 & 18 & 1 & 38 & 23 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 23 & 4 & 12 & 45 & 18 & 1 & 38 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 38 & 23 & 4 & 12 & 45 & 18 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 38 & 23 & 4 & 12 & 45 & 18 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 18 & 1 & 38 & 23 & 4 & 12 & 45 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 47 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 47 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 47 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 47 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 47 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 47 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 47
\end{pmatrix}.$$

The LLL-reduced Matrix related to M is

$$M' = \begin{pmatrix}
-1 & -1 & 0 & 0 & -1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\
1 & -1 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\
-1 & 0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\
0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\
-1 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\
-1 & 0 & -1 & 1 & -1 & 0 & 1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\
5 & 4 & 0 & -15 & 0 & 4 & 4 & 4 & -10 & 0 & 3 & 14 & -4 & -7 \\
-1 & 3 & 7 & 7 & -7 & -8 & -3 & -7 & -4 & -15 & -14 & -11 & 4 & 0 \\
8 & 7 & -7 & -7 & -4 & -1 & 4 & -14 & -14 & -11 & 4 & 0 & -8 & -4 \\
-1 & -15 & 0 & 4 & 4 & 3 & 4 & 0 & 3 & 14 & -3 & -7 & 4 & -11 \\
15 & 1 & -3 & -4 & -3 & -3 & 0 & -4 & -14 & 4 & 7 & -4 & 11 & 0 \\
4 & 5 & 4 & 4 & 1 & -14 & 0 & -4 & -7 & 4 & -11 & 0 & 4 & 14 \\
1 & -3 & -4 & -3 & -3 & 0 & 15 & -14 & 4 & 7 & -4 & 11 & 0 & -4
\end{pmatrix}$$

By denoting the row vectors of M' by $\mathbf{m}'_1, \cdots, \mathbf{m}'_{14}$ we will recognize that the shortest length in this lattice is $\|\mathbf{m}'_1\| = |\sqrt{6}|$. Furthermore, we have the vector

$$(\boldsymbol{f}, \boldsymbol{g}) = (1,1,1,-1,-1,0,0,0,1,1,0,-1,-1,0) = \mathbf{m}_3 + \mathbf{m}_6 - \mathbf{m}_1.$$

However, we want to find this vector or a vector related to a rotation $(x^k \boldsymbol{f}(x), x^k \boldsymbol{g}(x))$ analytically. Precisely, we are searching for short solutions

$$\mathbf{y} = \mathbf{x}M' \in \mathscr{T}(2d+1, 2d).$$

where $\mathbf{x}$ is an element of $\mathbb{Z}^{14}$. In addition, the vector $(y_1, \cdots, y_7)$ is in $\mathscr{T}(d+1, d)$ and the vector $(y_8, \cdots, y_{14})$ is in $\mathscr{T}(d, d)$ where $d = 1, 2$, or 3. Let

$$\mathbf{y} = \alpha_1 \mathbf{m}'_1 + \cdots + \alpha_{14} \mathbf{m}'_{14}.$$

We will search in the vectors where α_i is in $\{-1, 0, 1\}$ for $1 \leq i \leq 7$ and $\alpha_i = 0$ for $8 \leq i \leq 14$. To generate and save these vectors we will use the following code.

```
LinearCombinationsList = []
for i in range(-1,2):
    for j in range(-1,2):
        for k in range(-1,2):
            for l in range(-1,2):
                for m in range(-1,2):
                    for n in range(-1,2):
                        for o in range(-1,2):
                            LinearCombinationsList = LinearCombinationsList
                                    ... + [[i,j,k,l,m,n,o,0,0,0,0,0,0,0]]
```

```
LatticeShortVectors = []
for i in LinearCombinationsList:
    LatticeShortVectors.append(vector(i)*Lattice)
```

This list has $3^7 = 2187$ element. The maximum length of a vector that follows our criteria is $\sqrt{13}$ because it may contain exactly one zero coordinate. Furthermore, even if our vector has no zero coordinate, the least length of a bigger vector is $\sqrt{14}$. Thus we can use these facts to decrease the number of vectors that we are searching in.

```
RealyShortVectors = []
for i in range(len(LatticeShortVectors)):
    if (EuclideanLength(LatticeShortVectors[i])<4):
        RealyShortVectors.append(LatticeShortVectors[i])
```

In this set, we will have non-ternary elements which we may exclude.

```
S = {-1,1}
TernaryShortVectors =[]
for i in RealyShortVectors:
    if (set(i.coefficients()) == S):
        if(sum(i.coefficients()) ==1):
            TernaryShortVectors.append(i)
```

The sum of the coordinates of an element in $\mathscr{T}(2d+1,2d)$ is 1. Now we will recognize that we have only 14 elements left for searching.

```
TernaryShortVectors
        [(1, 1, -1, -1, 0, 0, 1, 1, 1, 0, -1, -1, 0, 0),
         (0, 1, -1, 1, 0, -1, 1, 1, 0, 0, 0, 0, 0, -1),
         (1, 1, 1, -1, -1, 0, 0, 0, 1, 1, 0, -1, -1, 0),
         (0, 1, 1, 1, -1, -1, 0, 0, 0, 1, 1, 0, -1, -1),
         (1, 0, -1, 1, 0, 1, -1, 0, 0, 0, -1, 1, 0, 0),
         (1, -1, -1, 0, 0, 1, 1, 1, 0, -1, -1, 0, 0, 1),
         (-1, 1, 0, 1, -1, 1, 0, 0, -1, 1, 0, 0, 0, 0),
         (1, 0, 1, -1, 1, 0, -1, -1, 1, 0, 0, 0, 0, 0),
         (0, 0, 1, 1, 1, -1, -1, -1, 0, 0, 1, 1, 0, -1),
         (1, -1, 1, 0, -1, 1, 0, 0, 0, 0, 0, 0, -1, 1),
         (-1, 1, 0, -1, 1, 0, 1, 0, 0, 0, 0, -1, 1, 0),
         (0, -1, 1, 0, 1, -1, 1, 0, 0, -1, 1, 0, 0, 0),
         (-1, 0, 0, 1, 1, 1, -1, -1, -1, 0, 0, 1, 1, 0),
         (-1, -1, 0, 0, 1, 1, 1, 0, -1, -1, 0, 0, 1, 1)]
```

Let $\mathbf{v}$ be a vector in the set `TernaryShortVectors`. These vectors represent polynomials as

$$\boldsymbol{f}_v(x) = \sum_{i=1}^{7} v_i x^{i-1} \quad \text{and}$$
$$\boldsymbol{g}_v(x) = \sum_{i=8}^{14} v_i x^{i-8}.$$

We want to check that $\boldsymbol{f}_v(x) \star \boldsymbol{h} = \boldsymbol{g}_v(x)$ for each of these vectors. We may apply

```
def Check_Possible_Solutions(List,H):
    V = []
    for i in List:
        V = list(i)
        f = 0
        for i in range(0,len(V)/2):
            f = f + V[i]*x^i
        f = RQ(f * H %Modulus)
        f = R(f.map_coefficients(lambda c : c.lift_centered(),ZZ))
        g = 0
        for i in range(len(V)/2,len(V)):
            g = g + V[i]*x^(i % floor(len(V)/2))
        if (g == f):
            print(V," is a possible solution.")
```

If we run the previous routine to our list of ternary short vectors, we will recognize that each of these vectors is a possible solution to our problem. Denote the vectors in this list as $\mathbf{v}_1, \cdots, \mathbf{v}_{14}$. In Table 2.2, we mention the vectors from this list which represent rotations of our private key.

Vector	Rotation of the Private Key
$\mathbf{v}_1$	$(x^6\boldsymbol{f}(x), x^6\mathbf{g}(x))$
$\mathbf{v}_3$	$(\boldsymbol{f}(x), \mathbf{g}(x))$
$\mathbf{v}_4$	$(x\boldsymbol{f}(x), x\mathbf{g}(x))$
$\mathbf{v}_6$	$(x^5\boldsymbol{f}(x), x^5\mathbf{g}(x))$
$\mathbf{v}_9$	$(x^2\boldsymbol{f}(x), x^2\mathbf{g}(x))$
$\mathbf{v}_{13}$	$(x^3\boldsymbol{f}(x), x^3\mathbf{g}(x))$
$\mathbf{v}_{14}$	$(x^4\boldsymbol{f}(x), x^4\mathbf{g}(x))$

Table 2.2: The Rotations of the Private Key $(\boldsymbol{f}(x), \boldsymbol{g}(x))$.

The parameter N in this example is very small such that we can apply the LLL-algorithm. In real-life scenarios with $N \geq 251$ [HSW05], we will obtain lattices of dimensions $2N \geq 502$. In the case of high dimensions, the LLL-algorithm cannot be applied [Jef14]. Thus,

NTRU is considered to be secure and it is implemented in several designs of nominated PQC public key cryptographic mechanisms [Ala+22].

Bibliography

[AA20] Hailiza Kamarulhaili, Arif Mandangan, and Muhammad Asyraf Asbullah. *A Security Upgrade on the GGH Lattice-based Cryptosystem*. 2020. https://www.researchgate.net/publication/343793639_A_Security_Upgrade_on_the_GGH_Lattice-based_Cryptosystem. Accessed 6.May.2024.

[Ala+22] Gorian Alagic et al. *Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. National Institute of Standards and Technology (NIST). 2022. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=934458. Accessed 6.May.2024.

[Axl24] SheldonAxler. *Linear Algebra Done Right*. Springer, 2024, pp. 354-369.

[Bab85] László Babai. *On Lovász Nearest Lattice Reduction and Lattice Point Problem*. 1985.https://link.springer.com/content/pdf/10.1007/BFb0023990.pdf. Accessed 4.May.2024.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. *Public-Key Cryptosystems from Lattice Reduction Problems In Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*. Springer, 1997, pp. 112-131.

[Gil15] Gilbert Baumslag, Benjamin Fine, Martin Kreuzer, and Gerhard Rosenberger. *A Course in Mathematical Cryptography*. DE GRUYTER. 2015. pp. 345-362.

[HSW05] Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. *Choosing Parameter Sets for NTRUEncrypt with NAEP and SVES-3. In: Topics in Cryptology – CT-RSA*. Springer. 2005. https://link.springer.com/chapter/10.1007/978-3-540-30574-3_10#citeas. Accessed 4.May.2024.

[Inf23] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik (BSI)). *BSI – Technical Guideline (Cryptographic Mechanisms: Recommendations and Key Lengths)*. BSI. 2023. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile&v=6. Accessed 4.May.2024.

[Jef14] Jeffrey Hoffstein, Jill Pipher, and Joseph H.Selverman. *An Introduction to Mathematical Cryptography*. Springer, 2014, pp. 373-454.

[Jud18] Thomas W. Judson. *Abstract Algebra Theory and Applications*. Orthogonal Publishing L3C, 2018, pp. 258-275.

[McA11] Alsdair McAndrew. *Introduction to Cryptography with Open-Source Software*. CRC Press, 2011, pp. 27-44.

[Ngu99] Phong Nguyen. *Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto '97*. In *Advances in Cryptology — CRYPTO' 99*. Springer, 1999, pp. 288-304.

[NIS16] NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. NIST. 2016. `https://csrc.nist.gov/csrc/media/projects/post-quantum-cryptography/documents/call-for-proposals-final-dec-2016.pdf`. Accessed 6.May.2024.

[Reg04] Oded Regev. *Lattices in Computer Science Course.* Tel Aviv University, 2004, `https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/`. Accessed 4.May.2024.

[SE16] J. J. Stapleton and W. Clay Epstein. *Security Without Obscurity A Guide to PKI Operations*. CRC Press, 2016, pp. 11-19.

[SM07] Mark Stamp and Richard M.Low. *Applied Cryptoanalysis*. John Wiley & Sons, Inc, 2007, pp. 293-304.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 6. Mai 2024