BACHELOR

# NTRU

## a lattice-based encryption system

de Wit, Bart

*Award date:*
2014

Technische Universiteit
Eindhoven
University of Technology

**Mathematics & Computer Science**

Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands

**Author**
Bart de Wit, 0741692

**Date**
November 22, 2014

# NTRU - a lattice-based encryption system

Technische Universiteit
**Eindhoven**
University of Technology

# Table of contents

**Title**
NTRU - a lattice-based encryption system

Technische Universiteit
**Eindhoven**
University of Technology

# Table of contents

**Title**
NTRU - a lattice-based encryption system

# 1 Introduction

**Abstract.** The NTRU cryptosystem introduced in [1] is a public key cryptosystem. In this thesis the NTRU system will be described from a perspective which is easier to understand. We will explain how and why NTRU works and we will use a lot of examples.

For a really long time RSA, which was introduced in 1978 [3], has been the go-to cryptosystem. But the first prototype for a Quantum Computer has already been built [6]. So this asks for a cryptosystem that can withstand a quantum algorithm. For this, Jeffrey Hoffstein, Jill Pipher and Joseph H. Silverman introduced NTRU [1]. For understanding their article, a good understanding of cryptography is needed. In this thesis we will show how the NTRU algorithm works and why it works. We will show a lot of examples and we will take very small steps. Additionally, we will discuss a few attacks on the NTRU cryptosystem. Some mathematical background is needed for understanding this. If you have a Bachelor degree in Mathematics, or if you are studying for one, you should be able to understand it.

There are two types of cryptography: Symmetric cryptography and Public key cryptography. Symmetric crypto uses the same key $k$ to encrypt and to decrypt messages. So when a person knows how to encrypt a message, he also knows how to decrypt it. This has its advantages as well as its disadvantages. A disadvantage of this is that the key $k$ has to be kept secret to the public, but has to made available to the other person who wants to have a conversation. The problem is that, in order to let that person know which key $k$ to use without anyone else hearing it. Public key crypto can be used to do this securely, because then the encryption key and decryption key differ. Thus the encryption key can be made public without letting anyone know how to decrypt the message, hence the name 'public key' cryptography.

Public key crypto works in a different manner. Let's say that a computer and a server want to have a secure conversation, like when you want to go shopping on the internet, there will happen a few things. First, the computer tells the server that he wants to communicate. After this, the server sends its public key $h$ to the computer. After this the computer needs to know if this public key is indeed the public key from the server, and not from someone impersonating the server. This is done with signatures. Once the computer knows this for sure, the computer will send a key $k_1$ (encrypted with the public key $h$) to the server. After this the computer and the server will both know $k_1$ and therefore can talk securely to each other by using Symmetric crypto, which is much more time efficient. A different computer contacting the same server, will use the same public key $h$ for the server, but a different key (say $k_2$) for the symmetric conversation. For a graphic display see Figure 1.1.

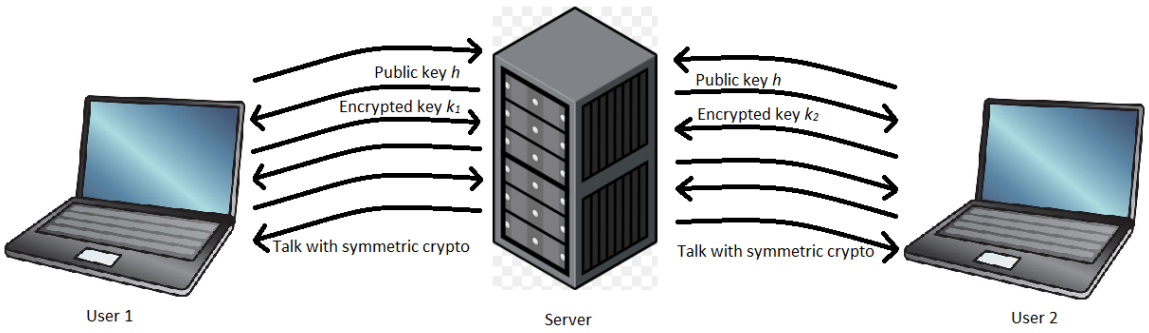


Figure 1.1: Two computers talking secure to the same server.

In this thesis we will talk about the public key $h$ and sending an encrypted message using this key and decrypting it, not about the networking protocol or symmetric cryptography.

# 2 Mathematical background

Like we said before, a little mathematical background is needed. In this chapter we will show some calculations that we will use all over the report.

## 2.1 A polynomial modulo a natural number

In this thesis we will use a lot of modulo calculations. $m \pmod p$ is just keeping the remainder of $m$ after division by $p$, so $15 \equiv 3 \pmod 4$. Now we will show how to take a polynomial modulo a natural number. To do this we have to calculate all the coefficients $F_i$ of a polynomial $F$ modulo this natural number. Just to refresh the memory. If we calculate $\pmod p$ we always find an integer from $0$ to $p-1$. For example $F = 13x^5 - 18x^4 + 5x^2 - x + 1$. Reducing $F$ modulo $5$ gives: $F \equiv 3x^5 + 2x^4 + 4x + 1 \pmod 5$. We can also reduce the coefficients to integers in $\left[-\lfloor \frac{q-1}{2} \rfloor, \lceil \frac{q-1}{2} \rceil\right]$. These integers are centered around $0$, then reducing $F$ modulo $5$ gives: $-2x^5 + 2x^4 - x + 1$.

## 2.2 A polynomial modulo a polynomial

We also need to know how to calculate modulo a polynomial. Here we use the same principles: we just keep the remainder after division by the modulus (a polynomial). For example: What is $2x^5 + 3x^4 + 8x^3 - x^2 + 3$ modulo $x^3 - 1$? (see the calculation below).

$$
\begin{array}{rrrl}
 & x^3 - 1 \;/\; 2x^5 + 3x^4 + 8x^3 - x^2 + 3 & \backslash & 2x^2 + 3x + 8 \\
\mathbf{2x^2}(x^3 - 1) = & \underline{2x^5 \qquad\qquad - 2x^2} & - & \\
 & 3x^4 + 8x^3 + x^2 + 3 & . & \\
\mathbf{3x}(x^3 - 1) = & \underline{3x^4 \qquad\quad - 3x} & - & \\
 & 8x^3 + x^2 + 3x + 3 & . & \\
\mathbf{8}(x^3 - 1) = & \underline{8x^3 \qquad\quad - 8} & - & \\
 & x^2 + 3x + 11 & . &
\end{array}
$$

so $2x^5 + 3x^4 + 8x^3 - x^2 + 3 \equiv x^2 + 3x + 11 \pmod{x^3 - 1}$.

Now we know that we can compute a polynomial $F$ modulo a number (say $p$) and modulo a polynomial (say $g(x)$). We denote this with $\pmod{g(x), p}$.

In more mathematical terms we work in the polynomial ring over $(\mathbb{Z}/p)[x]$, the ring with the coefficients reduced modulo $p$, modulo a polynomial. This ring is commonly denoted by $(\mathbb{Z}/p)[x]/g(x)$.

## 2.3 Example of computing an inverse modulo a polynomial and modulo $p$

Let $f = x^2 + x - 1$. We want to find the inverse of $f \pmod{(x^3 - 1), 3}$. We do this using the Extended Euclidean Algorithm. This algorithm uses two polynomials as input, $f$ and $g$, where $g$ is the modulus, in this case $x^3 - 1$, and has as output three polynomials, GCD$(f, g)$, $r$ and $s$, such that GCD$(f, g) = r \cdot g + s \cdot f \equiv s \cdot f \pmod{g}$. This all is calculated modulo $p$. In order for an inverse to exist, GCD$(f, g)$ has to be $1$ (for the calculation of this example, see below).

| # | scrap paper | polynomial | $r$ | $s$ | |
|---|---|---|---|---|---|
| 1 | | $x^3 - 1$ | 1 | 0 | |
| 2 | | $x^2 + x - 1$ | 0 | 1 | |
| 3 | $x^3 - 1 - x \cdot (x^2 + x - 1) =$ | $-x^2 + x - 1$ | 1 | $-x$ | subtract the $2^{nd}$ row $x$ times from the $1^{st}$ row |
| 4 | $x^2 + x - 1 + (-x^2 + x - 1) =$ | $2x - 2$ | 1 | $1 - x$ | add the $3^{rd}$ row the the $2^{nd}$ row |
| 5 | $-x^2 + x - 1 - x \cdot (2x - 2) =$ $-x^2 + x - 1 - 2x^2 + 2x =$ $-3x^2 + 3x - 1 \equiv -1$ | $-1$ | $1 - x$ | $x^2 - 2x$ | subtract the $4^{th}$ row x times from the $3^{rd}$ row |

So $-s = -x^2 + 2x$ is the inverse $\pmod{(x^3 - 1), 3}$ of $x^2 + x - 1$. Note that the $3^{rd}$ and $4^{th}$ row can be joined to make it one step.

Let us verify this by calculating $f \cdot (-s) \pmod{(x^3 - 1), 3}$. $f \cdot (-s) = (x^2 + x - 1) \cdot (-x^2 + 2x) = -x^4 + x^3 + 3x^2 - 2x \equiv 3x^2 - 3x + 1 \pmod{x^3 - 1} \equiv 1 \pmod{(x^3 - 1), 3}$.

## 2.4 Modulo multiplication

Until now $g$ and $p$ were very general, everything we described so far works independent of the shape of $g$ and $p$. Now we specialize the shape of $g$ to $x^N - 1$, which makes reductions modulo $g$ particularly easy.

We write $\circledast$ to denote multiplication modulo $x^N - 1$. This *star multiplication* is given explicitly as a cyclic convolution product:

$$F \circledast G = H, \text{ with } H_k = \sum_{i=0}^{k} F_i G_{k-i} + \sum_{i=k+1}^{N-1} F_i G_{N+k-i} = \sum_{i+j \equiv k \pmod N} F_i G_j. \qquad (2.1)$$

For example we could have $N = 3$, $F = F_0 + F_1 x + F_2 x^2$, $G = G_0 + G_1 x + G_2 x^2$. Then:

$$\begin{aligned} F \circledast G &= (F_0 + F_1 x + F_2 x^2)(G_0 + G_1 x + G_2 x^2) \\ &= F_0 G_0 + (F_0 G_1 + F_1 G_0) x + (F_0 G_2 + F_1 G_1 + F_2 G_0) x^2 + (F_1 G_2 + F_2 G_1) x^3 + F_2 G_2 x^4 \\ &\equiv F_0 G_0 + F_1 G_2 + F_2 G_1 + (F_0 G_1 + F_1 G_0 + F_2 G_2) x + (F_0 G_2 + F_1 G_1 + F_2 G_0) x^2 \pmod{x^3 - 1}. \end{aligned}$$

This last notation is the same as the last notation in (2.1) and can be found by dividing the second to last equation by $x^3 - 1$ with remainder.

When we do a multiplication modulo (say) $q$, we reduce the coefficients modulo $q$ after multiplication.

# 3 Description of the NTRU algorithm

We will use the same notation as in the paper of Jeffrey Hoffstein, Jill Pipher and Joseph H. Silverman, "NTRU: a ring-based public key cryptosystem." [1]. We use three integer parameters $(N, p, q)$ and we work in the ring: $\mathcal{R} := \mathbb{Z}[X]/(X^N - 1)$. An element $F \in \mathcal{R}$ will be written as a polynomial or a vector,

$$F = \sum_{i=0}^{N-1} F_i x^i = [F_0, F_1, \ldots, F_{N-1}]. \tag{3.1}$$

For example we could have:

$$F = 1 + 3x - 4x^3 = 1 + 3x + 0x^2 - 4x^3 + 0x^4 + 0x^5 + 0x^6 = [1, 3, 0, -4, 0, 0, 0], \text{ for } N = 7.$$

We define four sets of polynomials $(\mathcal{L}_f, \mathcal{L}_g, \mathcal{L}_\phi, \mathcal{L}_m) \subset \mathcal{R}$ with integer coefficients. $\mathcal{L}_f$ is the set of polynomials from which the private key comes. $\mathcal{L}_g$ is the set of polynomials from which part of the public key comes. $\mathcal{L}_\phi$ is the set of polynomials from which a polynomial is randomly chosen to help with the encryption. This is done so that the encryption is randomized. $\mathcal{L}_m$ is the set of polynomials from which the message can be chosen. These sets differ, because there are different restrictions on the functions of these sets. How these sets are constructed will be discussed in Chapter 4.2. $p$ and $q$ do not need to be prime, but they are required to be co-prime (so GCD$(p, q) = 1$), and $q \gg p$ ($q$ will always be much larger than $p$).

When we describe the algorithm, we will keep a running example to make it easier to follow. After each part of the algorithm we will update the running example with a new part. This running example will be computed in Mathematica [2].

## 3.1 Key generation

Now that we got the notation under control we can start with the key creation. Before anybody can send an encrypted message to Bob, he first needs to generate a key pair $(f, h)$ consisting of a public key $h$ and a private key $f$. These keys are needed to encrypt and decrypt the message. These keys are Bobs keys and the public key can be used by anyone. So Bob does not use different keys for everyone.

To generate his keys he randomly chooses two polynomials $f \in \mathcal{L}_f, g \in \mathcal{L}_g$. The polynomial $f$ must satisfy the requirement that it has inverses in $\mathcal{R}$ modulo $p$ and $q$. If we choose the parameters of $\mathcal{L}_f$ wisely (this wil be discussed in Section 4.2) this will be true for most $f$. It is not hard to compute these inverses. To do this we can use a modification of the Euclidean algorithm, see 2.3. We denote these inverses by $f_p$ and $f_q$. Bob uses the Euclidean algorithm to compute $f_p$ and $f_q$ such that:

$$f_p \circledast f \equiv 1 \pmod{p} \qquad \text{and} \qquad f_q \circledast f \equiv 1 \pmod{q}.$$

After doing this, he computes:

$$h \equiv f_q \circledast g \pmod{q}. \tag{3.2}$$

This is Bob's public key and has to be made public. His private key is the polynomial $f$ and has to be kept secret. It is smart to store the polynomial $f_p$ as well and keep is secret. $g$ and $f_q$ can be discarded. The next section shows how to send encrypted messages to Bob using $h$.

### 3.1.1 Running example

In this running example we will use $N = 11$, $p = 3$ and $q = 32$. We choose $f = x^{10} - x^9 + x^7 + x^4 - 1$ and $g = x^{10} + x^6 - x^3 - x^2 + x + 1$. This gives us:

$$\begin{aligned} f_p &= x^{10} + x^9 + 2x^8 + 2x^7 + 2x^5 + 2x^4 + 2x^2 + 1 \\ f_q &= 30x^{10} + 12x^9 + 8x^8 + 5x^7 + 28x^6 + 10x^5 + 19x^4 + 12x^3 + x^2 + 7x + 29. \end{aligned}$$

With $f_q$ and $g$ we compute:

$$h = 13x^{10} + 29x^9 + 20x^8 + 19x^7 + 9x^6 + 10x^5 + 13x^4 + 4x^3 + 30x^2 + 23x + 24.$$

### 3.1.2 Mathematica script

This script can be used to rebuild and play with the system.

```
(*First we define the parameters N, p and q. I use n instead of N \
```

```
because N is protected.*)
{n, p, q} = {11, 3, 32};
(*M is used to construct R = Z[X]/(X^N - 1)*)
M = x^n - 1;
(*f is the secret key of the receiver, fp is the inverse of f in R \
(mod p)*)
f = x^10 - x^9 + x^7 + x^4 - 1;
(*Choosing g*)
g = x^10 + x^6 - x^3 - x^2 + x + 1;
fp = PolynomialMod[
   Algebra`PolynomialPowerMod`PolynomialPowerMod[f, -1, x, M], p];
fq = PolynomialMod[
   Algebra`PolynomialPowerMod`PolynomialPowerMod[f, -1, x, M], q];
fp // TraditionalForm
fq // TraditionalForm
(*r1 and r2 are to check if fp and fq are indeed the inverses of f. \
In this case r1 and r2 have to be 1*)
r1 = PolynomialMod[fp*f, {M, p}];
r1 == 1
r2 = PolynomialMod[fq*f, {M, q}];
r2 == 1
(*Compute the public key h*)
h = PolynomialMod[fq*g, {M, q}];
h // TraditionalForm
```

## 3.2 Encryption

Let's say that Alice wants to send a secret message to her friend Bob. First Bob needs to tell Alice how to encrypt the message so he can decrypt it. First she needs to select the message from the plaintexts $\mathcal{L}_m$. In Section 4.2 we will show how $\mathcal{L}_m$ is constructed. Next she randomly chooses a polynomial $\phi \in \mathcal{L}_\phi$ and uses Bob's public encryption key $h$ to compute

$$c \equiv p\phi \circledast h + m \pmod{q}. \tag{3.3}$$

This is the encrypted message that she sends to Bob. $\phi$ is used to randomize the encryption and to hide the message. The message $m$ is visible in this expression, but because it is a star multiplication of functions there is no way to see the message $m$ that is hidden in $c$ once we use polynomial expressions (see the example below).

### 3.2.1 Running example

Alice wants to send message $m = x^{10} + 2x^9 + x^5 + x + 1$ to Bob. For the encryption she uses the public encryption key $h$ from Section 3.1.1. And she uses a randomly chosen polynomial $\phi \in \mathcal{L}_\phi$, let us use for example: $\phi = x^9 - x^7 + x^6 + x^3 - x - 1$. With these she computes $c = 28x^{10} + 4x^9 + 5x^8 + 7x^7 + 16x^6 + 31x^5 + 8x^4 + 3x^3 + 13x^2 + 20x + 31$. This is the message she sends to Bob.

### 3.2.2 Mathematica script

```
(*Choose the message m*)
m = x^10 + 2 x^9 + x^5 + x + 1;
(*Compute c*)
phi = x^9 - x^7 + x^6 + x^3 - x - 1;
c = PolynomialMod[p*PolynomialMod[phi*h, M] + m, q];
c // TraditionalForm
```

## 3.3 Decryption

Now Bob has received ciphertext $c$ from Alice. From this ciphertext $c$ he would like to compute message $m$ by using his private key $f$. To do this he first has to compute:

$$a \equiv f \circledast c \pmod{q}.$$

where he chooses the coefficients of $a$ in the interval $\left[-\lfloor \frac{q-1}{2} \rfloor, \lceil \frac{q-1}{2} \rceil\right]$. After this computation he can find the secret message $m$ by computing:

$$f_p \circledast a \pmod{p}.$$

### 3.3.1 Running example

With the message $c$ that Bob received from Alice he computes $a = 2x^{10} + 5x^9 + 31x^8 + 28x^7 + 27x^6 + 2x^5 + 11x^4 + 9x^3 + 2x^2 + 26x + 23$, using his private key. This polynomial does not have all its coefficients $\in [-15, 16]$. So he adjusts them to find $a = 2x^{10} + 5x^9 - x^8 - 4x^7 - 5x^6 + 2x^5 + 11x^4 + 9x^3 + 2x^2 - 6x - 9$. After multiplying $a$ with $f_p$ in modulo 3 he finds the hidden message $m = x^{10} + 2x^9 + x^5 + x + 1$ which is indeed the same secret message that Alice sent him.

### 3.3.2 Mathematica script

```
(*Compute a*)
```

```
a = PolynomialMod[f*c, {M, q}];
a // TraditionalForm
(*Adjust the coefficients*)
For[i = 0, i < n, i++,
 If[Coefficient[a, x, i] > q/2, a = a - q*x^i]]
a // TraditionalForm
(*Compute message m*)
me = PolynomialMod[fp*a, {M, p}];
me // TraditionalForm
```

## 3.4 How the decryption works

It is not easy to see that the decryption works and why it works. So we will go over the decryption step by step.

The polynomial $a$ has the following structure:

$$\begin{aligned} a &\equiv f \circledast c \\ &\equiv f \circledast (p\phi \circledast h + m) \pmod{q}, \quad \text{because of (3.3)} \\ &= f \circledast p\phi \circledast h + f \circledast m \pmod{q} \\ &= f \circledast p\phi \circledast f_q \circledast g + f \circledast m \pmod{q}, \quad \text{because of (3.2)} \\ &= p\phi \circledast g + f \circledast m \pmod{q}, \quad \text{because } f \circledast f_q \equiv 1 \pmod{q}. \end{aligned} \tag{3.4}$$

For appropriate parameter choices, see Section 4.2, we can ensure that (almost always) all of the coefficients of (3.4) lie between $-\lfloor \frac{q-1}{2} \rfloor$ and $\lceil \frac{q-1}{2} \rceil$, so $a$ does not change if its coefficients are reduced modulo $q$. This means that when Bob reduces the coefficients of $a \pmod{q}$ into the interval from $-\frac{q}{2}$ to $\frac{q}{2}$, he recovers exactly the polynomial

$$a = p\phi \circledast g + f \circledast m \quad \text{in} \quad \mathbb{Z}[X]/(X^N - 1). \tag{3.5}$$

Reducing $a$ modulo $p$ gives him the polynomial $f \circledast m \pmod{p}$, and,

$$f_p \circledast (f \circledast m) \pmod{p} \equiv m \pmod{p}. \tag{3.6}$$

Usually knowing a number modulo $q$ makes you non the wiser about that number modulo $p$. For example take $p = 3$ and $q = 5$. If $a \equiv 2 \pmod 5$, then $a$ could be $2 \pmod 3$,

because $a$ could be $2$. Or $a$ could be $1 \pmod 3$, because $a$ could be $7$. And $a$ could be $0 \pmod 3$, because $a$ could be $12$.

But if we know that $a$ is an integer between $-2$ and $2$, then we know in fact that $a = 2$, and the reduction of this integer value is unique modulo $3$, here $a \equiv 2 \pmod 3$.

This is the reason for choosing $q$ much larger than $p$.

# 4 Parameter selection

We gave a very general description of the NTRU algorithm. There are a lot of parameters to choose so the algorithm works almost always, is faster and is more secure. A few general parameters are $N$, $p$ and $q$. $N$ and $q$ are usually powers of $2$. And $p$ is usually a small odd number. This is to ensure that GCD$(p, q) = 1$ and $q \gg p$.

## 4.1 Notation and a norm estimate

We define the *width* of an element $F \in \mathcal{R}$ to be the difference between the smallest and largest coefficient of $F$:

$$|F|_\infty = \max_{0 \le i \le N-1} \{F_i\} - \min_{0 \le i \le N-1} \{F_i\}. \tag{4.1}$$

And we define $\bar{F} = \frac{1}{N} \sum_{i=0}^{N-1} F_i$ (the average of the coefficients) to define the following norm:

$$|F|_2 = \left( \sum_{1=0}^{N-1} (F_i - \bar{F})^2 \right)^{\frac{1}{2}}. \tag{4.2}$$

These norms differ a little bit from the $L^2$-norm and the $L^\infty$-norm. We do this because we calculate modulo $q$ and $p$, we know a lot about the width of the polynomials. These norms are useful for estimating certain parameters so the algorithm works better. Also note that the $2$-norm from above does not change if we add (say) $q$ to every entry of $F$. This is the reason for choosing this $2$-norm and $\infty$-norm.

Note that $|F|_\infty = 0 \Leftrightarrow |F|_2 = 0 \Leftrightarrow$ all the entries of $F$ are the same.

For example we will calculate $|F|_\infty$ and $|F|_2$ for $F = 10x^3 - 3x^2 + 5x + 8$. The largest coefficient of $F$ is 10, and the smallest is $-3$. Thus $|F|_\infty = 10 - (-3) = 13$.

The average of the coefficients is $\frac{1}{4}(10 - 3 + 5 + 8) = 5$. This gives us: $|F|_2 = \big((10 - 5)^2 + (-3 - 5)^2 + (5 - 5)^2 + (8 - 5)^2\big)^{\frac{1}{2}} = \sqrt{98} \approx 9.9$.

The following proposition was suggested by Don Coppersmith.

**Proposition.** For any $\varepsilon > 0$ there are constants $\gamma_1, \gamma_2 > 0$, depending on $\varepsilon$, such that for randomly chosen polynomials $F, G \in \mathcal{R}$, with probability greater than $1 - \varepsilon$ they satisfy:

$$\gamma_1 |F|_2 |G|_2 \leq |F \circledast G|_\infty \leq \gamma_2 |F|_2 |G|_2. \tag{4.3}$$

Of course this is not very helpful if for very small values of $\varepsilon$, $\gamma_1$ gets really small and $\gamma_2$ very large. However, it turns out that even for very small values of $\varepsilon$ and moderately large values of $N$ the ratio $\frac{\gamma_2}{\gamma_1}$ does not become very large.

Let us verify this.

We take $N = 64$ and $q = 128$. Then the coefficients of $F$ and $G$ are between $-\lfloor \frac{q-1}{2} \rfloor = -63$ and $\lceil \frac{q-1}{2} \rceil = 64$. This because our largest modulus is $q$. Now we calculate examples for $\gamma$. To calculate this we randomly chose $F$ and $G$ such that its coefficients are between $-63$ and $64$ and we calculate $|F|_2 |G|_2$ and $|F \circledast G|_\infty$. And we calculate $\gamma = \frac{|F \circledast G|_\infty}{|F|_2 |G|_2}$. If we do this often, lets say $1\,000\,000$ times and sort them, then we get $\gamma_1 \leq \gamma_2 \leq \cdots \leq \gamma_{1\,000\,000}$. Then $\gamma_1 |F|_2 |G|_2 \leq |F \circledast G|_\infty \leq \gamma_{1\,000\,000} |F|_2 |G|_2$ holds for all these randomly chosen $F$ and $G$.

Let us take a look at the following: For all but the examples with $\gamma_1 \ldots \gamma_{5\,000}$ we have that $\gamma_{5\,001} |F|_2 |G|_2 \leq |F \circledast G|_\infty$. And for all but the examples with $\gamma_{995\,000} \ldots \gamma_{1\,000\,000}$ we have that $|F \circledast G|_\infty \leq \gamma_{994\,999} |F|_2 |G|_2$. Thus for all but $10\,000$ out of $1\,000\,000$, i.e. for 99% it holds that $\gamma_{5\,001} |F|_2 |G|_2 \leq |F \circledast G|_\infty \leq \gamma_{994\,999} |F|_2 |G|_2$.

If we run this in Mathematica [2], we find: $\frac{\gamma_{1\,000\,000}}{\gamma_1} \approx 3.36$, and $\frac{\gamma_{994\,999}}{\gamma_{5\,001}} \approx 1.92$. For the Mathematica script see Chapter 6.

## 4.2 The sets of polynomials, $\mathcal{L}_f$, $\mathcal{L}_g$, $\mathcal{L}_\phi$ and $\mathcal{L}_m$

Like we said before, there are a lot of parameters to choose from to make the algorithm work better. Perhaps the easiest of these is the set of polynomials from which the message can be chosen, $\mathcal{L}_m$. Because of the mod $p$ calculation in 3.6, and assuming that $p$ is odd we require that:

$$\mathcal{L}_m := \left\{ m \in \mathcal{R} | \text{ All of the coefficients of } m \text{ lie beween } -\frac{1}{2}(p-1) \text{ and } \frac{1}{2}(p-1) \right\}. \tag{4.4}$$

Thus the coefficients of a polynomial $m \in \mathcal{L}_m$, and thus the message $m$, are not altered by the mod $p$ calculation.

For the other sample spaces we will use the following notation:

$$\mathcal{L}(d_1, d_2) := \left\{ F \in \mathcal{R} | F \text{ has } \begin{array}{l} d_1 \text{ coefficients equal to } 1, \\ d_2 \text{ coefficients equal to } -1, \\ \text{The remaining } (N - d_1 - d_2) \text{ coefficients are equal to } 0 \end{array} \right\} \quad (4.5)$$

For example, say $N = 5$ and $f \in \mathcal{L}(2, 1)$. Then $f$ could be $x^4 + 0x^3 - x^2 + 0x + 1$.

With this notation we choose three positive integers $d_f, d_g$ and $d_\phi$. With these we construct the following sets:

$$\mathcal{L}_f = \mathcal{L}(d_f, d_f - 1) \quad (4.6)$$
$$\mathcal{L}_g = \mathcal{L}(d_g, d_g) \quad (4.7)$$
$$\mathcal{L}_\phi = \mathcal{L}(d_\phi, d_\phi). \quad (4.8)$$

We chose coefficients $-1, 0$ and $1$ so the coefficients stay in control after a (star) multiplication and adding. We choose $\mathcal{L}_f$ not to be $\mathcal{L}(d_f, d_f)$, because a polynomial $f \in \mathcal{L}(d_f, d_f)$ satisfies $f(1) = 0$. This is because there are just as many one's and minus one's after filling in $x = 1$. A polynomial satisfying $f(1) = 0$ is not invertible modulo $x^N - 1$, because if $f(1) = 0$, then $f(x) = (x - 1)k(x)$, where $k(x) = \frac{f(x)}{x-1}$. And $x^N - 1 = (x - 1)(x^{N-1} + x^{N-2} + \cdots + x + 1)$. As we saw in Section 2.3 in order for $f$ to have an inverse, we need GCD$(x^N - 1, f(x))$ to be $1$. But GCD$((x - 1)(x^{N-1} + x^{N-2} + \cdots + 1), f(x)) =$ gcd$(x^N - 1, (x - 1)k(x))$ which is at least $x - 1$, or a multiple of $x - 1$. And we need $f$ to have an inverse in order for the algorithm to work, therefore we choose $\mathcal{L}_f = \mathcal{L}(d_f, d_f - 1)$

Now let us take a look at $|f|_2, |g|_2$ and $|\phi|_2$, for $f \in \mathcal{L}_f, g \in \mathcal{L}_g$ and $\phi \in \mathcal{L}_\phi$. Then we find:

$$|f|_2 = \sqrt{d_f(1 - \tfrac{1}{N})^2 + (d_f - 1)(-1 - \tfrac{1}{N})^2 + (N - 2d_f + 1)(0 - \tfrac{1}{N})^2} =$$
$$\sqrt{d_f - 2\tfrac{d_f}{N} + \tfrac{d_f}{N^2} + d_f + 2\tfrac{d_f}{N} + \tfrac{d_f}{N^2} - 1 - \tfrac{2}{N} - \tfrac{1}{N^2} + \tfrac{1}{N} - 2\tfrac{d_f}{N^2} + \tfrac{1}{N^2}} = \sqrt{2d_f - 1 - N^{-1}},$$

$$|g|_2 = \sqrt{2d_g}, \qquad |\phi|_2 = \sqrt{2d_\phi}.$$

By adjusting $d_f, d_g$ and $d_\phi$ we can be very sure that the decryption works and maintains various security levels (see Section 4.3).

## 4.3 Decryption criterion

Like we discussed in Section 3.4, we want $p\phi \circledast g + f \circledast m$ not to change when we calculate modulo $q$. So we want: $|p\phi \circledast g + f \circledast m|_\infty < \frac{q}{2}$.

Note that $|p\phi \circledast g + f \circledast m|_\infty \leq |p\phi \circledast g|_\infty + |f \circledast m|_\infty$.

So if we chose the parameters such that $|p\phi \circledast g|_\infty < \frac{q}{4}$ and $|f \circledast m|_\infty < \frac{q}{4}$, then $|p\phi \circledast g + f \circledast m|_\infty < \frac{q}{2}$.

From Proposition 4.3 we know that this will be true with probability $1 - \varepsilon$ if we chose:

$$|\phi|_2|g|_2 < \frac{q}{4p\gamma_2}, \qquad \text{and} \qquad |f|_2|m|_2 < \frac{q}{4\gamma_2}. \tag{4.9}$$

We calculated some different $\gamma_1$'s and $\gamma_2$'s. We calculated these the same way we did before, but now we only did it $2\,500$ times and took $\gamma_1$ to be the smallest and $\gamma_2$ to be the biggest. We started with $N = 40$, then $N = 50$ until $N = 490$. In Figure 4.1 it is clear to see that $\gamma_1$ and $\gamma_2$ get smaller and get closer to each other when $N$ gets larger.

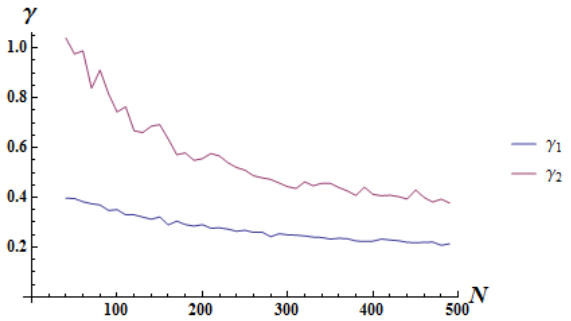


Figure 4.1: $\gamma_1$ and $\gamma_2$ by different dimensions $N$'s.

# 5 Attacks on NTRU

The secret message $m$ that Alice wants to send to Bob is a polynomial. But the message that Alice really sends is the polynomial $c = p\phi \circledast h + m \pmod q$. Because of the modulo $q$ every $N$ of the coefficients could be $q, 2q, \ldots$ bigger (or smaller) then it is was in $p\phi \circledast h + m$. There is no way to know how much, or which of the coefficients are altered. Also since $\phi$ is chosen randomly the attacker does not know it.

## 5.1 Brute force attack

If an attacker intercepts the message $c$, then he knows $(N, p, q), h$ and $c$. Knowing that $f \circledast h = g$ which has coefficients $-1$, $0$ and $1$ see (4.7) he could try all $f \in \mathcal{L}_f$ and take a look if $f \circledast h \pmod q$ has small coefficients. An attacker could also try all possible $g \in \mathcal{L}_g$ and calculate $g \circledast h^{-1} \pmod q = g \circledast (f_q \circledast g)^{-1} \pmod q = g \circledast g^{-1} \circledast f_q^{-1} \pmod q \equiv f \pmod q = f$, to see if it has small coefficients. He could also try to find the message directly. To do this he can try all $\phi \in \mathcal{L}_\phi$ and calculate $c - p\phi \circledast h \pmod q$. If he found the correct $\phi$, then he again should get a message with coefficients $-1$, $0$ and $1$, see (4.8).

In order to be secure against a brute force attack, the number of elements of $\mathcal{L}_f$, $\mathcal{L}_g$ and $\mathcal{L}_\phi$ have to be really large. We denote the number of elements in $\mathcal{L}_f$ by $\#\mathcal{L}_f$, and similarly $\#\mathcal{L}_g$ and $\#\mathcal{L}_\phi$. Now let us compute their values.

An element $f \in \mathcal{L}_f$ has the following structure in vector notation (3.1): $(\underbrace{-1, -1, \ldots, -1}_{d_f - 1}, \underbrace{0, \ldots, 0}_{N-2d_f+1}, \underbrace{1, \ldots, 1}_{d_f})$ in every order possible. So $\#\mathcal{L}_f = \binom{N}{d_f} \cdot \binom{N-d_f}{d_f-1} = \dfrac{N!}{d_f!(d_f-1)!(N-2d_f+1)!}$.

In the same manner we find: $\#\mathcal{L}_g = \dfrac{N!}{(d_g!)^2(N-2d_g)!}$ and $\#\mathcal{L}_\phi = \dfrac{N!}{(d_\phi!)^2(N-2d_\phi)!}$

To give an idea on how big the sets are we will use $N = 128, d_f = 15, d_g = 12$ and $d_\phi = 7$. This gives us $\#\mathcal{L}_f = 3\,624\,503\,100\,827\,074\,188\,373\,120\,924\,206\,720\,000 \approx 2^{121}$, $\#\mathcal{L}_g = 163\,188\,819\,629\,719\,810\,602\,829\,361\,208\,000 \approx 2^{107}$ and $\#\mathcal{L}_\phi = 5\,968\,388\,425\,947\,494\,976\,000 \approx 2^{72}$. Note that $\#\mathcal{L}_f, \#\mathcal{L}_g$ and $\#\mathcal{L}_\phi$ grow exponentially in $N$.

Doing one star multiplication for small $N$ does not take much time. In fact $100$ star multiplications can be done within a second on a laptop, but when $N$ gets larger the computations take quadratic in $N$ longer to compute. See Figure 5.1 for a graphic of the computation time in seconds for $N = 64$ until $512$.

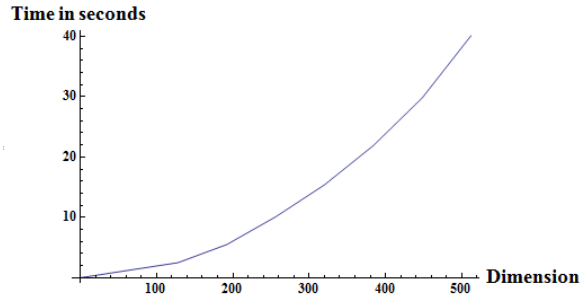


Figure 5.1: The amount of time it takes to do 100 star multiplications.

In order to deal with a brute force attack we need to make $N$ big enough.

## 5.2 Lattice attacks on NTRU

Now let us take a look at lattice attacks. We will see how a lattice attack on NTRU works, and if NTRU can survive a lattice attack.

### 5.2.1 A lattice

First, we will briefly explain what a lattice is, and what its problems are. For this we used [4].

Let $B$ be a matrix with column vectors $b_1, b_2, \ldots, b_k \in \mathbb{R}^n$. A lattice $L = B\mathbb{Z}^k = \left\{Bx \mid x \in \mathbb{Z}^k,\ B = [b_1, b_2, \ldots, b_k] \in \mathbb{R}^{n\times k}\right\}$ is a set of points in $\mathbb{R}^n$ obtained by taking integer combinations of the basis $b_1, b_2, \ldots, b_k$. For example we get the lattice shown in Figure 5.2 when we take $b_1 = (5, 1)^T$ and $b_2 = (4, 2)^T$.

The determinant of a lattice is defined by its generating matrix $B$. So $\det(L) = |\det(B)| = \sqrt{|\det(B^T B)|}$, where the last expression is used for not square matrices.

It is not hard to see that this lattice $L$ can also be generated by the basis $\{b_3, b_2\} = \left\{(-1, 1)^T, (4, 2)^T\right\}$ or many other bases. We are most interested in the basis $\{b_3, b_2\}$, because this is the shortest basis that generates $L$. This means that there is no vector $b \in L$ with $|b| < \max_i |b_i|$ such that $B$ with $b$ replacing any basis vector generates $L$.

The Shortest Vector Problem (SVP), is the problem of finding the shortest vector of a lattice $L$, i.e. it is to find the minimum of $|\ell|$, for some norm $|.|$, where $\ell \in L$ and $\ell \neq$ the zero vector. The problem to find short vectors is called approximate SVP.

In this case ($N = 2$) it is not really hard to solve SVP. This becomes harder when the two basis vectors are longer and differ just a bit from each other. When $N$ gets larger this problem gets significantly harder. There are algorithms like the LLL algorithm [5] that can find short

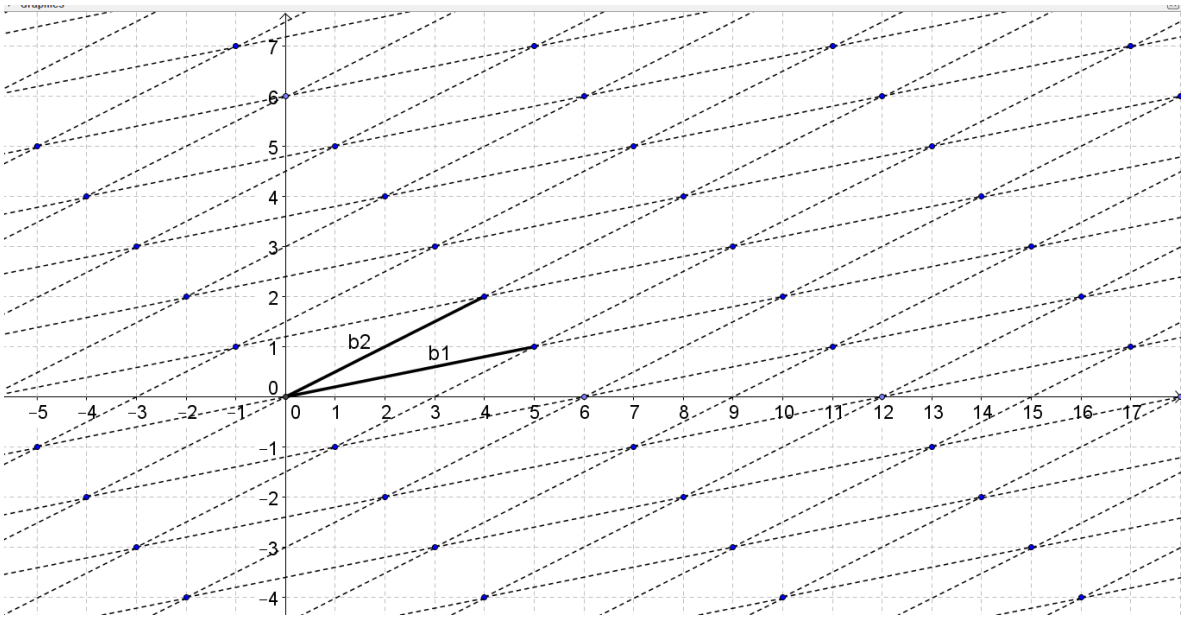


Figure 5.2: The lattice generated by $b_1 = (5, 1)^T$ and $b_2 = (4, 2)^T$.

vectors in polynomial time. The LLL algorithm can find a short vector that is at most $2^{\frac{N-1}{2}}$ times the length of the shortest vector in $L$. For properly constructed lattices LLL will not be able to find the shortest vector. In fact the chance to find the shortest vector is really small for arbitrary large $N$.

Another method is the Korkin-Zolotarev basis reduction method [7], which can find a short vector that is at most $\frac{\sqrt{N+3}}{2}$ times the length of the shortest vector in $L$ in polynomial time. And Kannan's SVP algorithm [8] later refined by Helfrich [9] with complexity $\leq N^{\frac{N}{2e}+(n)}$.

From now on we use $n = k$. There is an estimate of the length of the shortest vector of a lattice $L$. By the Gaussian heuristic, the expected length of the shortest vector in a random lattice of dimension $n$ and with determinant $D$ of $B$ lies in the interval:

$$\left( D^{\frac{1}{n}} \sqrt{\frac{n}{2\pi e}}, \quad D^{\frac{1}{n}} \sqrt{\frac{n}{\pi e}} \right). \tag{5.1}$$

A related problem is the closest vector problem (CVP), which is finding the closest lattice point to a given point. In Figure 5.3 we see a point $t$. It is clear to see that the vector $(7, 5)^T$ is the closest vector in the lattice to $t$, but this problem gets really hard when $N$ gets bigger.

SVP and CVP are connected problems. If we have some oracle that solves CVP for some

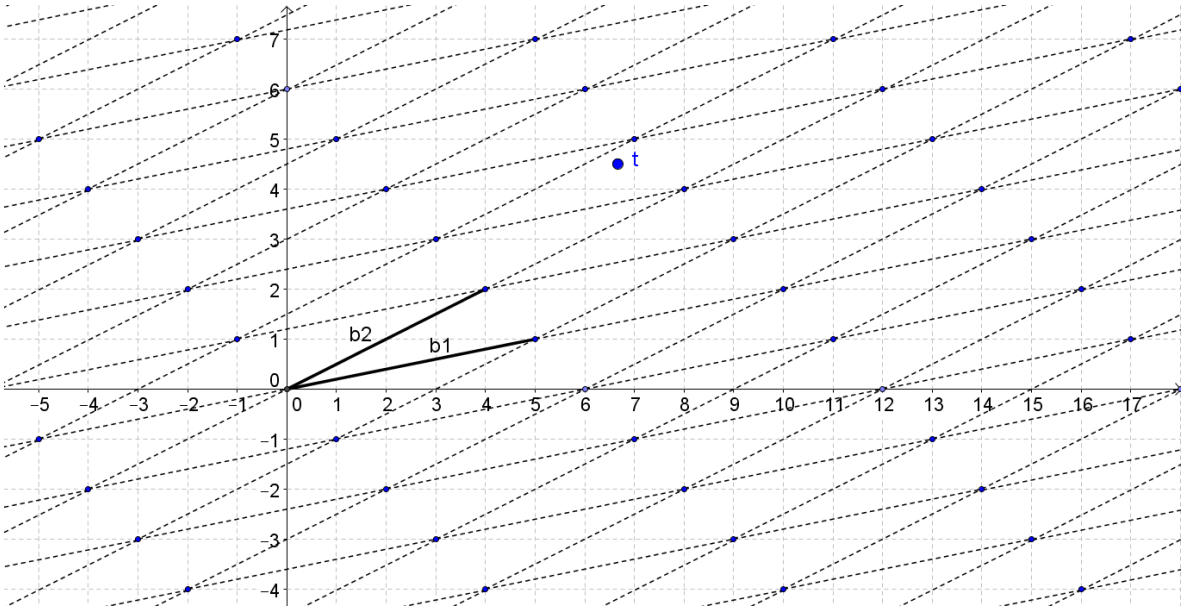


Figure 5.3: Closest vector problem to a target point $t$.

lattice $L$ with generator matrix $B$ and some target vector $t$, it is possible to solve SVP for this lattice. It is not possible to ask the oracle to solve CVP for $L$ and use the target vector $0$, this is because $0$ is always a lattice point. So CVP will return $|0-0|$ where $0$ is the $0$-vector. Let us make a new matrix from $B$ which we will call $B^i$. $B^i$ is the same as $B$ but the $i^{th}$ column is doubled. So $B^i = (b_1, b_2, \ldots, 2b_i, \ldots, b_N)$. When we ask the oracle to solve CVP for the lattice generated by $B^i$ and target vector $b_i$ it will return the closest vector point to $b_i$, let $x_i$ be the returned vector. The claim is that the shortest vector in the set $x_i - b_i$, for $i = 1, 2, \ldots, N$, is the solution to SVP. Solving SVP and CVP is NP-hard.

### 5.2.2 A lattice attack on $f$

As we saw in Section 5.2.1, we know that we can find a short vector of a lattice $L$. Now we will construct a lattice that is generated by a matrix with only known entries, and that contains $f$

Let us take a look at the lattice generated by the columns of the $2N$ by $2N$ matrix $B_f$ from (5.2), where $\alpha$ gets defined later.

$$B_f = \left(\begin{array}{cccc|cccc} \alpha & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & \alpha & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha & 0 & 0 & \cdots & 0 \\ \hline h_0 & h_{N-1} & \cdots & h_1 & q & 0 & \cdots & 0 \\ h_1 & h_0 & \cdots & h_2 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{N-1} & h_{N-2} & \cdots & h_0 & 0 & 0 & \cdots & q \end{array}\right). \tag{5.2}$$

This matrix only consists of known entries, because $h$ is the public key, $q$ is known and the attacker chooses $\alpha$ himself. Now we need to look if $f$ is in this lattice.

If we multiply the first column of $B_f$ with $f_0$, the second with $f_1$ until the $N^{th}$ column and add them, then we get the following:

$$\begin{pmatrix} \alpha f_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ f_0h_0 \\ f_0h_1 \\ \vdots \\ f_0h_{N-1} \end{pmatrix} + \begin{pmatrix} 0 \\ \alpha f_1 \\ 0 \\ \vdots \\ 0 \\ f_1h_{N-1} \\ f_1h_0 \\ \vdots \\ f_1h_{N-2} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \alpha f_2 \\ \vdots \\ 0 \\ f_2h_{N-2} \\ f_2h_{N-1} \\ \vdots \\ f_2h_{N-3} \end{pmatrix} + \cdots + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \alpha f_{N-1} \\ f_{N-1}h_1 \\ f_{N-1}h_2 \\ \vdots \\ f_{N-1}h_0 \end{pmatrix} = \begin{pmatrix} \alpha f_0 \\ \alpha f_1 \\ \alpha f_2 \\ \vdots \\ \alpha f_{N-1} \\ \sum_{i+j\equiv 0 \pmod N} f_ih_j \\ \sum_{i+j\equiv 1 \pmod N} f_ih_j \\ \vdots \\ \sum_{i+j\equiv N-1 \pmod N} f_ih_j \end{pmatrix}.$$

From (2.1) and the fact that $f \circledast h = g \pmod q$ we get that $\sum_{i+j\equiv 0 \pmod N} f_ih_j = g_0 \pmod q$, $\sum_{i+j\equiv 1 \pmod N} f_ih_j = g_1 \pmod q$, $\ldots$, $\sum_{i+j\equiv N-1 \pmod N} f_ih_j = g_{N-1} \pmod q$. Because of the vectors $b_{N+1}, \ldots, b_{2N}$, we can calculate modulo $q$ in the last $N$ entries of a vector in $L$. Because we can add and subtract $q$ as many times as we want from the last $N$ entries until we get an integer between $0$ and $q-1$.

So we get the vector: $\tau = (\alpha f, g)^T$. By this we mean the length-$2N$ vector consisting of the $N$ coefficients of $f$ multiplied by $\alpha$, followed by the $N$ coefficients of $g$. We can do this, because all the entries of $f$ are integers. Remember that $f$ is the private key, so its entries are unknown. So we can not find $f$ this way. But if $\tau$ is the shortest vector in the lattice $L$ then we can, in theory, find it using the LLL algorithm.

By using (5.1) and noticing that in our case we have $n = 2N$ and $D = \alpha^N q^N$, we know that the expected smallest length is larger (but not much larger) than:

$$s = (\alpha^N q^N)^{\frac{1}{2N}} \sqrt{\frac{2N}{2\pi e}} = \sqrt{\alpha q}\sqrt{\frac{N}{\pi e}} = \sqrt{\frac{N\alpha q}{\pi e}}. \tag{5.3}$$

We do not have a 2-norm for $\tau$, this because $\tau \notin \mathcal{R}$, but $\tau \in \mathcal{R}^2$. An element $T \in \mathcal{R}^2$ is composed of two elements ($T_1$ and $T_2$) $\in \mathcal{R}$. For that reason we choose $|T|_2 = \sqrt{|T_1|_2^2 + |T_2|_2^2}$. So we take the squared 2-norm from (4.2) for the first part and second part of $T$ and add them and then take the squareroot to get the 2-norm of $T$.

Let us take a look at the ratio $\frac{s}{|\tau|_2}$. $s \leq |\tau|_2$, because $s$ is a lower bound for the smallest vector of $L$, so we know that $\frac{s}{|\tau|_2} \leq 1$ at all times. But in order to locate the vector $\tau$ we want $\tau$ to be the shortest vector of $L$. To do this the attacker needs to choose $\alpha$ to maximize the ratio $\frac{s}{|\tau|_2}$, which is the same a maximizing the squared ratio $\frac{s^2}{|\tau|_2^2}$.

So $\alpha$ should maximize:

$$\frac{s^2}{|\tau|_2^2} = \frac{Nq}{\pi e} \cdot \frac{\alpha}{|\alpha f|_2^2 + |g|_2^2} = \frac{Nq}{\pi e} \cdot \frac{\alpha}{\alpha^2 |f|_2^2 + |g|_2^2} = \frac{Nq}{\pi e} \cdot (\alpha |f|_2^2 + \frac{1}{\alpha}|g|_2^2)^{-1}.$$

So we want $\alpha |f|_2^2 + \frac{1}{\alpha}|g|_2^2$ to be as small as possible. This is done by choosing $\alpha = \frac{|g|_2}{|f|_2}$. Note that $|g|_2$ and $|f|_2$ are public quantities.

When $\alpha$ is chosen this way, we define a constant $j_f$ which is the ratio of the length of the target vector to the length of the expected shortest vector. So:

$$j_f = \frac{|\tau|_2}{s} = \sqrt{\frac{2\pi e |f|_2 |g|_2}{Nq}}.$$

If $j_f$ is close to 1 (note that $j_f \geq 1$), then lattice reduction methods will have an easier time finding $\tau$. And when $\tau$ gets bigger so does $j_f$, which results in it being harder to find.

### Example of the matrix $B_f$

Now let us lake a look at the matrix $B_f$ that generates the lattice for the attack on the cryptosystem from the running example. Recall that $N = 11, p = 3, q = 32$ and $h = 13x^{10} + 29x^9 + 20x^8 + 19x^7 + 9x^6 + 10x^5 + 13x^4 + 4x^3 + 30x^2 + 23x + 24$. Let us take $d_f = 3$ and $d_g = 2$. This gives us $|f|_2 = \sqrt{\frac{54}{11}}$ and $|g|_2 = \sqrt{4} = 2$. Thus $\alpha = \sqrt{\frac{22}{27}}$

$B_f =$

$$\left(\begin{array}{cccccccccccc|cccccccccccc}
\sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{\frac{22}{27}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
24 & 13 & 29 & 20 & 19 & 9 & 10 & 13 & 4 & 30 & 23 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
23 & 24 & 13 & 29 & 20 & 19 & 9 & 10 & 13 & 4 & 30 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
30 & 23 & 24 & 13 & 29 & 20 & 19 & 9 & 10 & 13 & 4 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 30 & 23 & 24 & 13 & 29 & 20 & 19 & 9 & 10 & 13 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
13 & 4 & 30 & 23 & 24 & 13 & 29 & 20 & 19 & 9 & 10 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 13 & 4 & 30 & 23 & 24 & 13 & 29 & 20 & 19 & 9 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 \\
9 & 10 & 13 & 4 & 30 & 23 & 24 & 13 & 29 & 20 & 19 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 \\
19 & 9 & 10 & 13 & 4 & 30 & 23 & 24 & 13 & 29 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 \\
20 & 19 & 9 & 10 & 13 & 4 & 30 & 23 & 24 & 13 & 29 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 \\
29 & 20 & 19 & 9 & 10 & 13 & 4 & 30 & 23 & 24 & 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 \\
13 & 29 & 20 & 19 & 9 & 10 & 13 & 4 & 30 & 23 & 24 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0
\end{array}\right).$$

### 5.2.3 A lattice attack on $m$

A lattice attack may also be directed to a single message $m$. The associated lattice problem is very similar to that for $f$. Let us take a look at the lattice generated by the columns of the $2N+1$ by $2N+1$ matrix $B_m$ from (5.4), where $\beta$ gets defined later.

$$B_m = \left(\begin{array}{cccc|cccc|c}
\beta & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
0 & \beta & \cdots & 0 & 0 & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \cdots & \beta & 0 & 0 & \cdots & 0 & 0 \\
\hline
ph_0 & ph_{N-1} & \cdots & ph_1 & q & 0 & \cdots & 0 & c_0 \\
ph_1 & ph_0 & \cdots & ph_2 & 0 & q & \cdots & 0 & c_1 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
ph_{N-1} & ph_{N-2} & \cdots & ph_0 & 0 & 0 & \cdots & q & c_{N-1} \\
\hline
0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & 1
\end{array}\right). \tag{5.4}$$

Also this matrix only consists of known entries. The matrix looks like the matrix from the attack on $f$ (5.2), but the dimension is one larger. Also we use $ph$ instead of just $h$. We use it in the same cyclic manner.

If we multiply the first column of $B_m$ with $\phi_0$, the second column of $B_m$ by $\phi_1$ until the $N^{th}$ column with $\phi_{N-1}$, the $(N+1)^{th}$ column with $z_0$, the $(N+2)^{th}$ column with $z_1$ until the $2N^{th}$ column with $z_{n-1}$ and add them and subtract the last column from it (where $\phi_i$ is the $i^{th}$ coefficient of $\phi$ and the $z_i's$ are to reverse the modulo $q$ calculation) we get the following:

$$\begin{pmatrix} \beta\phi_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ p\phi_0 h_0 \\ p\phi_0 h_1 \\ \vdots \\ p\phi_0 h_{N-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \beta\phi_1 \\ 0 \\ \vdots \\ 0 \\ p\phi_1 h_{N-1} \\ p\phi_1 h_0 \\ \vdots \\ p\phi_1 h_{N-2} \\ 0 \end{pmatrix} + \cdots + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ \beta\phi_{N-1} \\ p\phi_{N-1} h_1 \\ p\phi_{N-1} h_2 \\ \vdots \\ p\phi_{N-1} h_0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ z_0 q \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ z_1 q \\ \vdots \\ 0 \\ 0 \end{pmatrix} + \cdots + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ z_{N-1} q \\ 0 \end{pmatrix}$$

$$- \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ c_0 \\ c_1 \\ \vdots \\ c_{N-1} \\ 1 \end{pmatrix} = \begin{pmatrix} \beta\phi_0 \\ \beta\phi_1 \\ \beta\phi_2 \\ \vdots \\ \beta\phi_{N-1} \\ \sum_{i+j\equiv 0 \pmod N} p\phi_i h_j + z_0 q - c_0 \\ \sum_{i+j\equiv 1 \pmod N} p\phi_i h_j + z_1 q - c_1 \\ \vdots \\ \sum_{i+j\equiv N-1 \pmod N} p\phi_i h_j - + z_{N-1} q - c_{N-1} \\ -1 \end{pmatrix} = \begin{pmatrix} \beta\phi_0 \\ \beta\phi_1 \\ \beta\phi_2 \\ \vdots \\ \beta\phi_{N-1} \\ -m_0 \\ -m_1 \\ \vdots \\ -m_{N-1} \\ -1 \end{pmatrix}.$$

From (3.3) we know that $c \equiv p\phi \circledast h + m \pmod q$, so $c_k = \sum_{i+j\equiv k \pmod N} p\phi_i h_j + m_k \pmod q$. This gives us: $\sum_{i+j\equiv k \pmod N} p\phi_i h_j - c_k + z_k q = -m_k$ for some integer $z_k$. Since $\phi_0, \phi_1, \ldots, \phi_{N-1}, z_0, \ldots, z_{N-1}$ and $-1$ are integers we now know that $\omega = (\beta\phi, -m, 1)^T \in L$, which will be a short vector.

Let $s_m$ be the length of the shortest vector in the lattice. Since $2N + 1 \approx 2N$ for $N$ large enough, we conclude:

$$s_m \approx \sqrt{\frac{N\beta q}{\pi e}}.$$

Since $\omega \notin \mathcal{R}^2$ we need to construct a $2-$norm for the length $2N+1$ vector $W = (W_1, W_2, W_3)^T$, where $W_3$ is the last coordinate of $W$. We define: $|W|_2 = \sqrt{|W_1|_2^2 + |W_2|_2^2 + |W_3|^2}$, note that $W_3 \in \mathbb{R}$. So we find $|\omega|_2^2 = \sqrt{|\beta\phi|_2^2 + |m|_2^2 + (1-1)^2} = \sqrt{\beta^2|\phi|_2^2 + |m|_2^2}$.

Let us calculate the value of $\beta$. To do this, we again take a look to the ratio $\frac{s_m^2}{|\omega|_2^2} = \frac{qN}{\pi e} \cdot \frac{\beta}{\beta^2|\phi|_2^2 + |m|_2^2}$.

So $\beta$ should maximize $\frac{\beta}{\beta^2|\phi|_2^2+|m|_2^2} = (\beta|\phi|_2^2 + \frac{1}{\beta}|m|_2^2)^{-1}$. This is done by choosing $\beta = \frac{|m|_2}{|\phi|_2}$, for the same reason as before.
We define:

$$j_m = \frac{|\omega|_2}{s_m} = \sqrt{\frac{2\pi e|m|_2|\phi|_2}{Nq}}.$$

In order to make the attacks on $f$ and $m$ equally difficult, the designer should take:

$$\begin{aligned} j_h &\approx j_m \\ \Longrightarrow \sqrt{\frac{2\pi e|f|_2|g|_2}{Nq}} &\approx \sqrt{\frac{2\pi e|m|_2|\phi|_2}{Nq}} \\ \Longrightarrow |f|_2|g|_2 &\approx |m|_2|\phi|_2. \end{aligned} \tag{5.5}$$

Let us take a look at what this means in the case of $p = 3$. In this case an average message $m$ consists of $\frac{N}{3}$ entries, each $-1$, $0$ and $1$. This gives us: $|m|_2 \approx \sqrt{\frac{2N}{3}}$. As seen in Section 4.2 $|\phi|_2 = \sqrt{2d_\phi}$. So we want to set $|f|_2|g|_2 \approx \sqrt{\frac{4}{3}Nd_\phi}$. From (4.9) we know that we should have $|f|_2|m|_2 < \frac{q}{4\gamma_2}$. So we want $|f|_2\sqrt{\frac{2N}{3}} < \frac{q}{4\gamma_2}$, which gives us $|f|_2 < \frac{q}{\sqrt{\frac{2N}{3}}4\gamma_2}$. So we should choose $d_f < \frac{q^2}{\sqrt{\frac{4N}{3}}16\gamma_2^2} + \frac{1}{2} + \frac{1}{2N}$, and choose $|g|_2 \approx \frac{\sqrt{\frac{4}{3}Nd_\phi}}{\sqrt{2d_f-1-N^{-1}}} = \sqrt{\frac{4Nd_\phi}{3(2d_f-1-N^{-1})}}$

**Suggested parameters**

An example of suitable suggested parameters in [1] are: $(N, p, q) = (107, 3, 64)$ and $(d_f, d_g, d_\phi) = (15, 12, 5)$. Let us verify if the above holds. First we check if $|f|_2|g|_2 \approx |m|_2|\phi|_2$. $|f|_2|g|_2 = \sqrt{2\cdot 15 - 1 - \frac{1}{107}} \cdot \sqrt{2\cdot 12} \approx 26.38$ and $|m|_2|\phi|_2 = \sqrt{\frac{2\cdot 107}{3}} \cdot \sqrt{2\cdot 5} \approx 26.71$, so $26.38 \approx 26.71$, which is true. Then we check if $d_f$: $15 < \frac{64^2}{\sqrt{\frac{107\cdot 4}{3}}16\cdot 1^2} + \frac{1}{2} + \frac{1}{107} \approx 21.43$ which again is true. So we finally check $|g|_2$: $\sqrt{24} \approx \frac{4*107*5}{3(30-1-\frac{1}{107})}$ which gives us: $4.90 \approx 4.96$, which is true.

Other suggested parameters from [1] are: $(N, p, q) = (167, 3, 128)$ with $(d_f, d_g, d_\phi) = (61, 20, 18)$ and $(N, p, q) = (503, 3, 256)$ with $(d_f, d_g, d_\phi) = (216, 72, 55)$.

### 5.2.4 A lattice attack on $m$ using CVP

Let us take a look at the lattice generated by the $2N$ by $2N$ matrix $B_{mc}$ from (5.6).

$$B_{mc} = \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 0 \\ \hline ph_0 & ph_{N-1} & \cdots & ph_1 & q & 0 & \cdots & 0 \\ ph_1 & ph_0 & \cdots & ph_2 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ ph_{N-1} & ph_{N-2} & \cdots & ph_0 & 0 & 0 & \cdots & q \end{array}\right). \tag{5.6}$$

In the same manner as in Section 5.2.3 an attacker knows that the vector from (5.7) is in the lattice. Which is close to the $2N-$length vector $(0,c)^T$, namely with distance $\sqrt{|\phi|_2^2+|m|_2^2}$.

$$\begin{pmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{N-1} \\ \sum_{i+j\equiv 0 \pmod N} p\phi_i h_j + z_0 q \\ \sum_{i+j\equiv 1 \pmod N} p\phi_i h_j + z_1 q \\ \vdots \\ \sum_{i+j\equiv N-1 \pmod N} p\phi_i h_j + z_{N-1} q \end{pmatrix} = \begin{pmatrix} \phi_0 \\ \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{N-1} \\ c_0 - m_0 \\ c_1 - m_1 \\ \vdots \\ c_{N-1} - m_{N-1} \end{pmatrix}. \tag{5.7}$$

Subtracting $(0,c)^T$ from (5.7) leads to the length$-2N$ vector $(\phi,-m)^T$ and thus gives the attacker $m$. But (5.7) has to be the closest vector in the lattice to $(0,c)^T$. Let us define $s_{mc}$ to be the length of the shortest vector of the lattice generated by $B_{mc}$. If $(0,c)^T$ has distance less then $\frac{1}{2}s_{mc}$ to (5.7), then we know that (5.7) is the closest vector. So we want to know if the following inequality holds for large $N$:

$$\begin{aligned} \sqrt{|\phi|_2^2+|m|_2^2} &< \frac{1}{2}\sqrt{\frac{Nq}{\pi e}} \\ \Longrightarrow \sqrt{2d_\phi + \frac{2}{3}N} &< \sqrt{\frac{Nq}{4\pi e}} \\ \Longrightarrow 2d_\phi + \frac{2}{3}N &< \frac{Nq}{4\pi e} \\ \Longrightarrow d_\phi &< \frac{Nq}{8\pi e} - \frac{1}{3}N. \end{aligned} \tag{5.8}$$

For large $N$ and $q$ the inequality holds easily. This means that (5.7) is the closest vector point to $(0, c)^T$. And thus an attacker is able to find $m$ using CVP if there is an exact CVP solver.

$$B_f = \left(\begin{array}{ccccc|ccccc} \sqrt{\frac{7}{10}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{\frac{7}{10}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{\frac{7}{10}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{\frac{7}{10}} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{\frac{7}{10}} & 0 & 0 & 0 & 0 & 0 \\ \hline 1 & 7 & 4 & 6 & 14 & 16 & 0 & 0 & 0 & 0 \\ 14 & 1 & 7 & 4 & 6 & 0 & 16 & 0 & 0 & 0 \\ 6 & 14 & 1 & 7 & 4 & 0 & 0 & 16 & 0 & 0 \\ 4 & 6 & 14 & 1 & 7 & 0 & 0 & 0 & 16 & 0 \\ 7 & 4 & 6 & 14 & 1 & 0 & 0 & 0 & 0 & 16 \end{array}\right).$$

# 6 Appendix

This program is used in Section 4.1 to compute the $\gamma's$

```
(*The number of runs*)
number = 1000000;
(*Construct an array y that keeps track of the gammas*)
y = ConstantArray[0, number];
(*Defining N*)
n = 64;
For[k = 0, k < number, k++,
 (*Constructing randomly chosen F and G and calculate F(bar) and \
G(bar)*)
 FF = RandomInteger[{-63, 64}, n];
 F = 0;
 f = 0;
 GG = RandomInteger[{-63, 64}, n];
 G = 0;
 g = 0;
 FFGG = ConstantArray[0, n];
 fgem = Total[FF]/n;
 ggem = Total[GG]/n;
 For[i = 1, i < n + 1, i++,
  f = f + (FF[[i]] - fgem)^2 ;
  g = g + (GG[[i]] - ggem)^2
  ];
 nf = (f)^0.5;
 ng = (g)^0.5;
 (*Make the array's of F and G into a polynomials*)
 For[i = 1, i < n + 1, i++,
  F = F + FF[[i]]*x^(i - 1);
  G = G + GG[[i]]*x^(i - 1)
  ];
 (*Calculate F star G and its infinity norm*)
 FG = PolynomialMod[F*G, x^n - 1];
 FFGG = CoefficientList[FG, x];
 nfg = Max[FFGG] - Min[FFGG];
 (*Calculate the gamma and add it to the array*)
 yy = nfg/(nf*ng);
 y[[k + 1]] = yy;
 ]
```

```
(*Sort y such that gamma(1) \[LessEqual] gamma(2) \[LessEqual] ... \
\[LessEqual] gamma(number)*)
sorted = Sort[y];
(*Calculate the ratio's*)
sorted[[number]]/sorted[[1]]
sorted[[0.995*number - 1]]/sorted[[0.005*number + 1]]
```

# 7 References


1. Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman. "NTRU: a ring-based public key cryptosystem." MR 1726077. Pages 267-288 in: Joe P. Buhler (editor). Algorithmic number theory. Proceedings of the 3rd international symposium (ANTS-III) held at Reed College, Portland, OR, June 21-25, 1998. Lecture Notes in Computer Science 1423. Springer. ISBN 3-540-64657-4. MR 2000g:11002.
2. Mathematica: `http://www.wolfram.com/mathematica/`
3. R.L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public key cryptosystems, Communications of the ACM 21 (1978), 120-126
4. Daniele Micciancio Foundations of Security Analysis and Design VI, UC San Diego, Lecture Notes in Computer Science Volume 6858, 2011, pp 185-210
5. A.K. Lenstra, H.W. Lenstra, L. Lovász, Factoring polynomials with polynomial coefficients, Math. Annalen 261 (1982), 515-534
6. http://dwave.wordpress.com/2011/05/11/learning-to-program-the-d-wave-one/
7. J. C. LAGARIAS, H. W. LENSTRA, JR. and C. P. SCHNORR. KORKIN-ZOLOTAREV BASES AND SUCCESSIVE MINIMA OF A LATTICE AND ITS RECIPROCAL LATTICE, COMBINATORICA 10 (4) (1990), pp 333-348
8. R. Kannan. Improved algorithms for integer programming and related lattice problems. In Proceedings of the 15th Symposium on the Theory of Computing (STOC 1983), pages 99-108. ACM Press, 1983
9. B. Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. Theor. Comput. Science, 41: 125-139, 1985.