Solving NTRU Challenges Using the New Progressive BKZ Library

Mårtensson, Erik

2016

Document Version:
Publisher's PDF, also known as Version of record

Solving NTRU Challenges Using the New Progressive BKZ Library

Erik Mårtensson

Department of Electrical and Information Technology
Lund University

Advisors: Professor Thomas Johansson and Qian Guo

2016-08-24

Printed in Sweden
E-huset, Lund, 2016

Abstract

NTRU is a public-key cryptosystem, where the underlying mathematical problem is currently safe against large-scale quantum computer attacks. The system is not as well investigated, as for example RSA and the company behind NTRU has created the NTRU Challenges, to remedy this. These challenges consist of 27 different public keys of increasing size, where the task in each challenge is to calculate (something similar to) the private key. The goal of this thesis was to examine different attacks against the NTRU Challenges and solve as many challenges as possible. By lattice reduction attacks, using a recently published new progressive BKZ algorithm, the first five challenges were solved, while the current biggest solved challenge by any researcher is challenge number seven.

Keywords: NTRU Challenge, Progressive BKZ , BDD, Enumeration, SVP

Acknowledgements

I would like to thank my main supervisor Professor Thomas Johansson for all his help and feedback during the project, for introducing me to the lattice-based cryptography area and last but not least for pointing out the necessity of solving a small problem correctly before tackling a big problem.

I would like to thank my assistant supervisor Qian Guo for all his help and feedback during the project and for introducing me to the progressive BKZ algorithm that turned out to become the focus of this thesis.

I would like to thank all the researchers that patiently answered my many questions during this project, with special thanks to Yoshinori Aono, Léo Ducas and Zhenfei Zhang for answering questions regarding the progressive BKZ algorithm and library, the BKZ plus BDD enumeration strategy for attacking NTRU and the NTRU Challenges respectively.

I would like to thank Erik Jonsson for helping with (at least for me) tricky installations and other technical difficulties.

I would like to thank to thank Måns Jarlskog for aiding in creating scripts that made the simulations much more automated.

Without the help of the aforementioned people, this thesis would not have been possible.

Erik Mårtensson

Table of Contents

List of Algorithms

Chapter 1

Goals and Overview

1.1 Goals

NTRU is a public-key cryptosystem, where the underlying mathematical problem is currently safe against large-scale quantum computer attacks. The system is not as well investigated, as the older and more established public-key cryptosystems. To combat this the company behind NTRU has created the NTRU Challenges, to test the strength of NTRU. These challenges consist of 27 different public keys of different sizes, where the task for each challenge is to, given the public key, calculate (something similar to) the private key.

One goal of this thesis is to investigate what different types of attacks there are against the NTRU Cryptosystem. Another goal is to develop lattice reduction based software specialised at attacking NTRU, using available open-source software. A final goal is to use these attacks to solve as many of the NTRU Challenges as possible.

1.2 Overview

Starting with Chapter 2, we introduce the reader to cryptography in general and in particular public-key cryptography, giving RSA as a classic example. In Chapter 3 we introduce the NTRU and the NTRU Challenges, and connect them to the SVP problem in lattices. Then in Chapter 4 we explain the original BKZ algorithm and the concept of pruned enumeration. In Chapter 5 we explain how extreme pruning and some other tricks gave rise to BKZ 2.0. Thereafter, in Chapter 6, we explain how several more optimizations created the new progressive BKZ algorithm. In Chapter 7 we explain the method of solving the NTRU Challenges, using the progressive BKZ algorithm in conjunction with some more specific tricks, and in Chapter 8 we go through the implementation of this method. Finally we go through the simulation results in Chapter 9 and discuss these in Chapter 10.

Appendix A illustrates the process of lattice reduction attacks by breaking NTRU using a toy sized key. For convenience, Appendix B contains a list of the most important acronyms of this thesis. For a popular scientific summary of the thesis, written in Swedish, see Appendix C.

Chapter 2
Preliminaries

In cryptography, a basic problem is the following. Alice is sending a secret message to Bob over an insecure channel. The enemy Eve can listen to the channel and Alice must thus encrypt the message in a way, such that Bob, but not Eve, can decrypt it[1]. This can be done in two different ways, symmetric key cryptography and public-key cryptography.

A historic example, perhaps the most famous example of encrypting messages, is the Caesar cipher, in which each letter of a message is shifted a certain number number of positions in the alphabet (if a letter gets shifted beyong 'z' the shift starts over at 'a' again). Caesar himself famously shifted the letters 3 steps, but any number of steps between 1 and 25 works equally well.

2.1 Symmetric Key Cryptography

In symmetric key cryptography Alice and Bob has decided on a common secret key k, from a set K of possible keys, beforehand through a secure channel. Every such key defines an invertible function f_k from a set M of plaintexts to a set of ciphertexts C.

In the Caesar cipher above, for example, the shared key is the number k of steps to shift each letter. Representing the letters of the English alphabets as numbers between 0 and 25 the function f_k for encrypting each single letter m is

$$c = f_k(m) = m + k \pmod{26},$$

and obviously the inverse function to decrypt each letter c of the ciphertext is

$$m = f_k^{-1}(c) = c - k \pmod{26}.$$

This cryptosystem is of course trivial to break. The currently most used symmetric cryptosystem is the Advanced Encryption standard (AES) [29]. It is currently

[1] To avoid unnecessarily complicated sentences such as "The receiver of the secret message distributes the public key to the sender of the message", placeholder names are used. The names Alice, Bob and Eve are the standard names for the sender of and receiver of the message, and the enemy trying to decipher it respectively. In more complicated scenarios a longer list of standard names are used.

able to safely and quickly encrypt plaintexts. AES and other symmetric key cryptosystems do have one big problem though. How do Alice and Bob safely establish the shared key in the first place? One elegant way of solving this problem is to establish the shared key using public-key cryptography, discussed in Section 2.2.

2.2 Public-key Cryptography

In public-key cryptography (also known as asymmetric key cryptography in contrast to symmetric key cryptography from Section 2.1) Bob uses two keys. One public key, available to anyone, with which anyone can encrypt a message and send it to Bob. One private key, only available to Bob, which is needed to decrypt messages, encrypted by the public key.

Before we can have a look at a specific public-key cryptosystem, we need the concepts of a one-way function and a trapdoor one-way function. A one-way function can informally be defined in the following way.

Definition 2.1. *(One-way function). A one-way function f is a function from a set X to a set Y, such that for all $x \in X$ it is easy to compute $f(x)$, but for essentially all $y \in Y$ it is computationally infeasible to find an $x \in X$ such that $f(x) = y$.*

Here computationally infeasible means that it takes impractically much time using available computing power. Depending on the patience and computers this limit can vary greatly. A one-way trapdoor function can be defined informally like this.

Definition 2.2. *(Trapdoor one-way function). A trapdoor one-way function f is a one-way function defined as above, but containing so called trapdoor information T. If and only if one has access to T, given a value $y \in Y$, it is easy to find an $x \in X$ such that $f(x) = y$.*

The idea of public-key cryptography is, that for Alice to encrypt the message corresponds to calculating a value of a one-way trapdoor function, which is always easy. Only Bob has access to the trapdoor information, and thus only he can decrypt the message, corresponding to inverting the value Alice computed.

Below follows a description of the first public-key cryptosystem, Rivest-Shamir-Adleman (RSA), named after its inventors Ron Rivest, Adi Shamir and Leonard Adleman, publicly described in 1977 [19][2].

2.3 Description of RSA

RSA is based on the fact that factoring a product of two big, unknown, prime numbers is computationally difficult. Before we can explain RSA we need definitions of two numbers being relatively prime and Euler's totient function.

[2]It should be mentioned that Clifford Christopher Cocks independently developed an equivalent system in 1973, working at the United Kingdom Government Communications Headquarters. The idea was classified and became publicly known first in 1997.

Definition 2.3. *(Relatively prime). Two integers a and b are called relatively prime if the only positive integer that divides both a and b is 1.*

Definition 2.4. *(Euler's totient function). Given a positive integer n, Euler's totient function, denoted as $\phi(n)$, counts the number of positive integers less than n that are relatively prime to n.*

As an example, if $n = pq$, where p and q are prime numbers, then

$$\phi(n) = \phi(p)\phi(q) = (p-1)(q-1).$$

2.3.1 Public Key

As part of the public key Bob uses a big composite number n, a factor of two (to everybody except Bob) unknown prime numbers p and q. He also uses a number e such that e and $\phi(n)$ are relatively prime.

2.3.2 Private Key

One part of the private key is the two prime numbers p and q such that $n = p \cdot q$. Another part of the private key is $\phi(n) = \phi(p) \cdot \phi(q) = (p-1)(q-1)$. The final part of the private key is $d = e^{-1} \pmod{\phi(n)}$. Since e and $\phi(n)$ are relatively prime such a value is guaranteed to exist due to Bezout's identity. The value can be calculated using the extended Euclidean algorithm.

2.3.3 Encryption

To send a message M, encoded as a number between 0 and $n-1$. Alice uses the private key to calculate

$$C = M^e \pmod{n}. \tag{2.1}$$

To speed up this calculation Alice can for example use exponentation by squaring, so that only roughly $\log e$ multiplications are needed[3].

2.3.4 Decryption

Given an encrypted number $C \in [0, n-1]$, Bob decrypts it by calculating

$$C^d \pmod{n} = (M^e)^d \pmod{n} = M^{ed} \pmod{n} = M^1 \pmod{n} = M. \tag{2.2}$$

Below follows a proof that this calulation always gives the original plaintext number.

[3]There are of course even faster methods for efficient exponentiation, but this method is more than half as fast as the fastest method possible.

Proof 2.1. *If p is a prime number and p does not divide the number a, then Fermat's little theorem states that*

$$a^{p-1} \equiv 1 \pmod{p}.$$

We want to show that $M^{ed} = M \pmod{p \cdot q}$. By construction $ed = 1 \pmod{\phi(n)}$, which means that there is an integer h such that

$$ed - 1 = h \cdot \phi(n) = h \cdot (p-1) \cdot (q-1).$$

According to the Chinese remainder theorem, to show that M and M^{ed} are congruent modulo $p \cdot q$, it is sufficient to show that the numbers are congruent to p and q separately. Let us show this for p, the proof for q is of course analogous. We divide the proof into two parts. If $M = 0 \pmod{p}$ then M^{ed} is a multiple of p and thus $M^{ed} = 0 = M \pmod{p}$. Otherwise, since p does not divide M, using Fermat's little theorem we get

$$M^{ed} = M^{ed-1} \cdot M = M^{h \cdot (p-1) \cdot (q-1)} \cdot M =$$
$$\left(M^{p-1}\right)^{h \cdot (q-1)} \cdot M = 1^{h \cdot (q-1)} \cdot M = M \pmod{p}.$$

□

2.3.5 Example of RSA

The following small example of how RSA works is from the the original RSA paper [19, p. 10]. Bob's public key is $(n, e) = (2773, 17)$. Bob also knows that

$$n = p \cdot q = 47 \cdot 59.$$

Next

$$\phi(n) = \phi(p) \cdot \phi(q) = \phi(47) \cdot \phi(59) = 46 \cdot 58 = 2668.$$

Finally

$$d = e^{-1} \pmod{\phi(n)} = 17^{-1} \pmod{2668} = 157.$$

Thus Bob's private key is $(p, q, d, \phi(n)) = (47, 59, 157, 2668)$. If Alice wants to encrypt the message 920 she calculates

$$C = M^e \pmod{n} = 920^{17} \pmod{2773} = 948.$$

Receiving this encrypted message Bob decrypts it by calculating

$$C^d \pmod{n} = 948^{157} \pmod{2773} = 920 = M.$$

2.3.6 Attacks on RSA

Knowing any of the numbers in the private key makes calculating the other parts trivial. It is believed, but not known, that there is no faster way of calculating all these numbers than to first factor n. It should be mentioned that when implementing the RSA system in practise, a lot of aspects outside the basic description above, such as hashing of the message, padding schemes and so on, has to be considered.

A classical computer stores data in bits, that can only take the value 0 or 1. A classical computer using n bits can be in 2^n different states. A quantum computer on the other hand uses quantum bits, or qubits in short. This means that a quantum computer using n qubits can be in a superposition of the 2^n different states.

In 1994 Peter Shor developed a new algorithm for factoring integers [24]. The algorithm consists of first reducing the factoring problem to the problem of finding the period of a certain function, and then finding the period of that function. The first step can be done in polynomial time on a classical computer (with respect to the number of digits in the number to factor). The second step is slow on a normal computer, but can be done in polynomial time on a quantum computer.

Currently, quantum computers are small and limited in the number of qubits. The largest number factored by a quantum computer is just $56153 = 233 \cdot 241$ [6], using only 4 qubits. This number is of course trivial to factor on a classical computer too. However, if there exists large scale quantum computers in the future, then RSA is broken. To combat this other public-key cryptosystems than RSA are needed. One such public-key system is the subject of Chapter 3.

Chapter 3
Introduction

3.1 The NTRU Cryptosystem

In 1996 Hoffstein, Pipher and Silverman developed a public-key cryptosystem called Nth Degree Truncated Polynomial (NTRU)[1][11].

3.1.1 Description of the NTRU system

In mathematical terms, polynomials in the polynomial ring

$$R[x]/(x^N - 1),$$

are used for plaintexts, ciphertexts, public and private keys etc., in NTRU. Here the underlying ring R is the finite field $\mathbb{Z}_n$, for some integer $n \geq 2$. Consider two such polynomials

$$a(x) = a_{N-1}x^{N-1} + \cdots + a_1x + a_0,\ b(x) = b_{N-1}x^{N-1} + \cdots + b_1x + b_0.$$

Adding these polynomials is done elementwise the usual way (remember to reduce the elements modulo n). The product of a and b is

$$(a * b)(x) = \sum_{k=0}^{N-1} \left(\sum_{\substack{0 \leq i,j \leq N-1 \\ i+j \equiv k \pmod N}} a_i \cdot b_j \right) x^k, \tag{3.1}$$

where the dot multiplication refers to the usual multiplication of two elements in $\mathbb{Z}_n$. Below follows how Alice sends a secret message to Bob and how Bob decrypts it.

[1] It should be mentioned that there are alternative definitions for the acronym like "Number Theorists aRe Us" and "Number Theory Research Unit"

System Parameters

The NTRU cryptosystem has three integer parameters, N, p and q. The above mentioned N has to be a prime number[2]. The p is a small number, almost always equal to 3. The description below assumes that $p = 3$. Changing the value p in the description below would essentially just change the coefficients in the message polynomial to integers in the set $[-p/2, p/2]$. Finally q is a number relatively prime to p and much bigger than p.

Key Generation

Bob randomly chooses two private key polynomials f and g from the polynomial ring above with coefficients in $\{-1, 0, 1\}$[3]. Also f must have an inverse modulo p, called f_p with coefficients in $\mathbb{Z}_p$ and an inverse modulo q, called f_q with coefficients in $\mathbb{Z}_q$. In other words, these polynomials must satisfy

$$f * f_q \equiv 1,\ f * f_p \equiv 1,$$

in $\mathbb{Z}_q/(x^N - 1)$ and $\mathbb{Z}_p/(x^N - 1)$ respectively. As the public key polynomial he chooses

$$h = p \cdot f_q * g \pmod{q}, \tag{3.2}$$

where the modulo operation is applied elementwise and the order of the operations does not effect the result.

Encryption

Alice puts her plaintext message in a polynomial m with coefficients in $\{-1, 0, 1\}$. She also picks a random polynomial r with small coefficients (not necesarily limited to the set $\{-1, 0, 1\}$). Using Bob's public key she encrypts the message as

$$e = r * h + m \pmod{q}. \tag{3.3}$$

Decryption

Using his private key f, Bob first calculates

$$a = f * e \pmod{q} = f * (r * h + m) \pmod{q} =$$
$$= f * (p \cdot r * f_q * g + m) \pmod{q} = p \cdot r * g + f * m \pmod{q}.$$

All the coefficients in a are shifted to the interval $[-q/2, q/2]$. This guarantees that $p \cdot r * g + f * m$ has coefficients in $[-q/2, q/2]$, since m, g, r and f all have

[2] As is shown in [9] letting N be composite significantly reduces the security of the NTRU system.

[3] There are several forms of these polynomials. The original NTRU system used binary coefficients and the NTRU challenge uses a more complex form of f. The point of this section is to explain how NTRU works without going too deep into all the details.

small coefficients. Reducing modulo q does thus not change the coefficients. Then he computes

$$b = a \pmod p = p \cdot r * g + f * m \pmod p = f * m \pmod p.$$

Finally he computes

$$c = f_p * b \pmod p = f_p * f * m \pmod p = m.$$

3.1.2 Example

The following is a toy example of this whole process from [30]. In this example $n = 11$, $p = 3$ and $q = 32$. Bob's private key is the polynomials

$$f(x) = -1 + x + x^2 - x^4 + x^6 - x^9 + x^{10},$$
$$g(x) = -1 + x^2 + x^3 + x^5 - x^8 - x^{10}.$$

Using the polynomial extended Euclidean algorithm the inverses of f modulo q and modulo p are calculated as

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

This creates the public key

$$h(x) = 3 \cdot f_q(x) * g(x) \pmod{32} = 8 - 7x - 10x^2 - 12x^3 + 12x^4 - 8x^5 + $$
$$+15x^6 - 13x^7 + 12x^8 - 13x^9 + 16x^{10} \pmod{32}.$$

Let us say Alice wants to send the message $[-1, 0, 0, 1, -1, 0, 0, 0, -1, 1, 1]$ corresponding to the polynomial

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

using the random polynomial

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

This results in the encrypted message

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

Having received this encrypted message from Alice, Bob first calculates

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

using the fact that he knows the private key polynomial $f(x)$. Then he calculates

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

Finally he gets

$$c(x) = f_p(x) * b(x) \pmod 3 = -1 + x^3 - x^4 - x^8 + x^9 + x^{10} \pmod 3,$$

which corresponds to the original message $[-1, 0, 0, 1, -1, 0, 0, 0, -1, 1, 1]$.

3.1.3 Attacks on the NTRU system

Brute-force

The most obvious way of attacking the NTRU system is by brute force. For a given format of the set of possible private keys f, Eve can test each such possible key to see if it is the actual private key. For a candidate private key f', she can calculate $f' * h$. For the correct key f this calculation gives g, which is a polynomial with very small coefficients. For a randomly chosen polynomial r this calculation typically gives a polynomial also containing large coefficients. It should be mentioned that getting a polynomial with small coefficients does not guarantee that the correct private polynomial f was found. For the polynomial $f_k = x^k f$ the calculation gives

$$f_k * h = x^k * f * h = x^k * g,$$

which is a polynomial also containing the correct coefficients, but circularly right shifted k steps. To check that a potential candidate polynomial f' is correct Eve tries to decrypt a message she herself has encrypted. To protect against brute force attacks the private keys has to be picked from a sufficiently large set, making these attacks computationally infeasible.

Meet-In-The-Middle Attack

Andrew Odlyzko originally described a meet-in-the-middle attack against NTRU private keys. This short description of the attack is from [18]. While this type of attack can be done in the more general case it is easiest to explain with binary keys. The attack uses the fact that any circular shift of the actual private key f generates the same coefficients (only shifted) when multiplied by h. Thus Eve to begin with only has to test polynomials f' of a certain form. Suppose also, for simplicity of explanation, that the system has an even size N and that f has an even number d_f of coefficients with the value 1 and $N - d_f$ coefficients with the value 0. Modifying the attack for prime system sizes N and odd values d_f is trivial.

The attack uses the fact that no matter how the private key f looks, it has a rotation such that there are $d_f/2$ ones in the first $N/2$ coefficients and $d_f/2$ ones in

the last $N/2$ coefficients. It is enough to test keys on this format and once a candidate f' appears for which $f' * h$ gives a polynomial with small coefficients, then all rotations of f' are tried until the actual private key f is found. This strategy makes the the amount of private key candidates to test much fewer than in the brute-force attack. To protect against this type of attack the set of possible private keys has to be expanded.

Lattice Reduction Attacks and Hybrid Attacks

The next type of attacks is lattice reduction attacks. They need quite a lot of explaining theory, which will be provided in Chapters 4-7. Another type of attacks is hybrids between meet-in-the-middle and lattice reduction attacks. They will not be explained in this thesis, but the interested reader is referred to the original hybrid attack paper [12].

3.2 Introduction to Lattice Reduction

This section will first introduce some notation and then introduce lattices and how lattices and NTRU are connected. This theory will be needed in Chapters 4-7, where lattice reduction attacks against NTRU are explained.

3.2.1 Notation

Given $d \in \mathbb{R}$ let $\lceil d \rceil$, $\lfloor d \rfloor$ and $\lceil d \rfloor$ denote the rounding of d upwards, downwards and to the closest integer respectively.

Let $\mathbb{R}^n$ denote the Euclidean space. Let bold letters denote row vectors. Given a polynomial $p(x) = p_0 + p_1 x + \cdots + p_{n-1}x^{n-1}$, the vector $\mathbf{p}$ refers to the row vector $(p_0, p_1, \ldots, p_{n-1})$. For two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, define the inner product and norm as

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \sum_{i=0}^{n-1} x_i \cdot y_i,$$
$$||\boldsymbol{x}|| = \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle} = \sqrt{\sum_{i=0}^{n-1} {x_i}^2}.$$

A set of vectors $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}\}$ in $\mathbb{R}^n$ are linearly independent if and only if the equation

$$\sum_{i=1}^{m} \lambda_i \mathbf{x_i} = \mathbf{0},$$

only has the solution $\lambda_1 = \lambda_2 = \cdots = \lambda_m = 0$. The (row) rank of a matrix in $M \in \mathbb{R}^{m \times n}$ is the maximum number of linearly independent (row) vectors of M. Given a set of vectors $\mathbf{B} = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_m}\}$ in $\mathbb{R}^n$, we define the span as

$$span(\mathbf{B}) = \{\mathbf{c_1 b_1} + \mathbf{c_2 b_2} + \cdots + \mathbf{c_m b_m} | \mathbf{c_i} \in \mathbb{R}\}.$$

Given a subset $E \subset \mathbb{R}^n$, denote its orthogonal complement as

$$E^{\perp} = \{\mathbf{x} \in \mathbb{R}^n : \langle \mathbf{x}, \mathbf{y} \rangle = 0, \text{ for all } \mathbf{y} \in E\}.$$

Now define the orthogonal projection of $\mathbb{R}^n$ on $E^{\perp}$ as the unique linear map $\pi_E : \mathbb{R}^n \to E^{\perp}$, such that

$$\pi_E(\mathbf{x}) = \mathbf{x}, \text{ for all } x \in E^{\perp},$$
$$\pi_E(\mathbf{x}) = \mathbf{0}, \text{ for all } x \in E.$$

For $s > 0$ the gamma function is defined as

$$\Gamma(s) = \int_0^{\infty} t^{s-1} e^{-t} dt.$$

Let $B_n(R)$ denote the n-dimensional Euclidean ball of radius R, or in other words the set

$$B_n(R) = \left\{ (x_1, \ldots, x_n) \in \mathbb{R}^n : \sum_{i=1}^{n} x_i^2 \leq R^2 \right\}.$$

Let $V_n(R)$ denote its volume, which satisfies

$$V_n(R) = R^n \cdot \frac{\pi^{n/2}}{\Gamma(n/2+1)}.$$

Similarly, let $S^{n-1}(R)$ denote the n-dimensional sphere of radius R. When the radius of V_n or S^{n-1} is omitted $R = 1$ is implied.

3.2.2 Definition of Lattices and Lattice Notation

We start by defining a lattice.

Definition 3.1. *(Lattice). A lattice $L \subseteq \mathbb{R}^n$ is a discrete, additive subgroup of $(\mathbb{R}^n, +)$.*

By discrete we mean that every element in L has a neighbourhood containing only that point. Next we define the rank of a lattice and what a lattice basis is.

Definition 3.2. *(Rank). The rank of a lattice L is the maximum number of linearly independent vectors in L.*

Definition 3.3. *(Basis). Let L be a lattice in $\mathbb{R}^m$ and let $\mathbf{B} = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$ be a set of linearly independent vectors in L. If for any vector $\mathbf{v} \in L$ there exists a unique set of integers $\{x_1, x_2, \ldots x_n\}$, such that $\mathbf{v} = x_1\mathbf{b_1} + x_2\mathbf{b_2} + \cdots + x_n\mathbf{b_n}$, then $\mathbf{B}$ is a basis of L. L is said to be spanned by $\mathbf{B}$.*

To indicate that a lattice L can be spanned by a basis $\mathbf{B}$, we denote it $L(\mathbf{B})$. A change of lattice basis corresponds to multiplying the current basis with a unimodular matrix[4]. This does not change the absolute value of the determinant of the basis. Thus we can define the volume of a lattice in the following way.

Definition 3.4. *(Volume). The volume of a lattice L is the absolute value of the determinant of any basis of L.*

This is of course the same as the hypervolume of the parallelepiped spanned by the basis vectors. One of the most important concepts in the theses is the following.

Definition 3.5. *(Shortest vector). The norm of the shortest (non-zero) vector of a lattice L is denoted* $\lambda_1(L) = \min_{\mathbf{v}\in L, \mathbf{v}\neq \mathbf{0}} ||\mathbf{v}||$.

The Shortest Vector Problem (SVP) is the problem of finding a vector in the lattice of length $\lambda_1(L)$. A related problem is the Closest Vector Problem (CVP), where the task is to find the lattice vector closest to a certain target vector. SVP is thus a special case of CVP where the target vector is the zero vector.

Given a lattice L and a "nice" set S, the Gaussian Heuristic (GH) predicts that the number of points in $S \cap L$ is about $vol(S)/vol(L)$. If the GH is true for a lattice we expect (heuristically, not in the probabilistic sense) that $\lambda_1(L) \approx GH(L) = (vol(L)/V_n(1))^{1/n}$. Next, we need definitions of sublattices.

Definition 3.6. *(Sublattice). Let L be a lattice in* $\mathbb{R}^m$*. A sublattice of L is a lattice* L' *in* $\mathbb{R}^m$*, included in L.*

Definition 3.7. *(Primitive sublattice). A sublattice* L' *of L is called primitive if and only if for any basis* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_r}\}$ *of* L' *there exists* $\{\mathbf{b_{r+1}}, \mathbf{b_{r+2}}, \ldots, \mathbf{b_n}\} \in \mathbf{L}$*, such that* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$ *is a basis of L.*

Given a lattice $L(\mathbf{B})$ with $\mathbf{B} = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$ and $\mathbf{B_r} = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_r}\}$, then $L(\mathbf{B_r})$ is a primitive sublattice of $L(\mathbf{B})$. Finally we need projective sublattices.

Lemma 3.1. *(Projective sublattice). Let L be a full-rank lattice in* $\mathbb{R}^n$ *and let* L' *be an r-rank primitive sublattice of L, where* $1 \leq r \leq n$*. Let* $\pi_{L'}$ *denote the orthogonal projection over* $span(L')^{\perp}$*. Then* $\pi_{L'}(L)$ *is a lattice of rank* $n - r$*.*

Proof 3.1. *Proof. Suppose* $\mathbf{B_r} = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_r}\}$ *is a basis of* L'*, which can be completed to a basis* $\mathbf{B} = \mathbf{B}_r \cup \{\mathbf{b_{r+1}}, \ldots, \mathbf{b_n}\}$ *of L. Then* $\pi_{L'}(L)$ *is the lattice generated by the basis* $\{\pi_{L'}(\mathbf{b}_{r+1}), \ldots, \pi_{L'}(\mathbf{b}_n)\}$*, where* $\pi_{L'}(\mathbf{b_i})$ *are linearly independent vectors for all i, such that* $r < i \leq n$*.* □

Given an implied basis $\mathbf{B} = \{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$ the projection $\pi_{L(\mathbf{B}_{i-1})}$ is henceforth denoted π_i for convenience of notation and for the sake of completeness we let π_1 denote the identity map. Let us also use $\mathbf{B}_{[i,j]}$ to denote the basis $\{\pi_i(\mathbf{b_i}), \ldots, \mathbf{ß_i}(\mathbf{b_j})\}$ and use $L_{[i,j]}$ to denote the lattice spanned by $\mathbf{B}_{[i,j]}$, where $1 \leq i \leq j \leq n$.

[4]A unimodular matrix is an integer matrix with the determinant equal to 1 or -1. An integer matrix has an integer matrix as inverse if and only if it is unimodular. This demand corresponds to the determinant being non-zero for a real (complex) matrix.

3.2.3 Connection Between Lattices and NTRU

Consider an NTRU cryptosystem with parameters (n, q, p), private keys f, g and a public key of the format

$$h = p \cdot f^{-1} * g \pmod{q} \Leftrightarrow p^{-1} \cdot h = f^{-1} * g \pmod{q}.$$

Let us denote $h' = p^{-1} \cdot h$ and rewrite the equation;

$$h' = f^{-1} * g \pmod{q} \Leftrightarrow f * h' = g \pmod{q}.$$

There exists an integer polynomial λ such that

$$f * h' + q \cdot \lambda = g.$$

On matrix form this can be written as

$$\begin{pmatrix} \mathbf{f} & \boldsymbol{\lambda} \end{pmatrix} \begin{pmatrix} I & H' \\ 0 & qI \end{pmatrix} = \begin{pmatrix} \mathbf{f} & \mathbf{g} \end{pmatrix}, \tag{3.4}$$

where the element on row i and column j in H' is $h'_{ij} = h'_{j-i \pmod{n}}$, if the first row and column are numbered as 0. Let L be the lattice generated by the rows of this matrix. We call this lattice the NTRU lattice. By construction the private key (f, g) is a vector in L. The determinant of L is q^N. By the GH the expected norm of the smallest vector in L (with dimension n and determinant q^N) is

$$\sqrt{\frac{Nq}{\pi e}}. \tag{3.5}$$

The norm of $(\mathbf{f}, \mathbf{g})$ is typically much less than this limit. Thus the task of finding the private key $(\mathbf{f}, \mathbf{g})$ can be solved by searching for the shortest vectors in L. This is typically done using lattice reduction followed by vector enumeration. How this is done is explained in Chapters 4-6. How to exploit the structure of the NTRU lattice to speed up this process is explained in Chapter 7. A small example from scratch of how the NTRU lattice is created and how the private key is found in this lattice, using lattice reduction, can be found in Appendix A.

3.3 The NTRU Challenges

The NTRU system has not undergone as much cryptanalytic testing as for example the more famous RSA. To combat this, Security Innovation, the company behind NTRU, issued the so called NTRU Challenge [13]. The challenge consists of 27 different NTRU systems with public key h and some associated system parameters, where the size of the public key ranges from $N = 107$ up to $N = 401$. The task for each system is to find the associated private key $(\mathbf{f}, \mathbf{g})$ or a similar vector. The public key has the format $h = f^{-1} * g \pmod{q}$[5]. The term similar vector refers to any vector $(\mathbf{f}', \mathbf{g}')$ in the corresponding NTRU lattice that has

[5]Notice that the factor p is not a part of the definition of h. This is due to the fact that if it would be there the first step of generating the NTRU lattice would be to multiply h by p^{-1} to get rid of the factor p anyways, as explained in Subsection 3.2.3.

a norm less than the GH, rounded upwards, in other words any vector $(\mathbf{f}', \mathbf{g}')$ satisfying

$$\|(\mathbf{f}', \mathbf{g}')\| < \left\lceil \sqrt{\frac{Nq}{\pi e}} \right\rceil. \tag{3.6}$$

The first correct submission for each problem receives a monetary prize. Each of the first 11 problems gives a prize of \$1000 and the last 17 problems gives \$5000 each. Beyond just the private key $(\mathbf{f}', \mathbf{g}')$, a short explanation of how the private key was calculated, has to be provided.

3.3.1 Format of the Challenges

Each challenge has a problem size N, the parameter q being equal to 2^n, where either $n = 9$, $n = 10$ or $n = 11$ and the implied value $p = 3$. Each challenge also has another 4 associated integer parameters; d_1, d_2, d_3 and d_g. The private key g has $d_g + 1$ coefficients equal to 1, d_g coefficients equal to -1 and the rest equal to 0. The private key f has the format

$$f = 1 + p \cdot F,$$

where

$$F = f_1 * f_2 + f_3.$$

Here each polynomial f_i has d_i coefficients equal to 1, d_i coefficients equal to -1 and the rest equal 0 for $i = 1, 2, 3$.

3.3.2 Effects of the Format

For practical application this set of parameters has some advantages. First of all $f^{-1} \pmod p = f_p = 1$ and thus does not have to be calculated. Secondly we have

$$f(1) = 1 + p \cdot (f_1(1) * f_2(1) + f_3(1)) = 1 + p \cdot (0 \cdot 0 + 0) = 1.$$

This makes the probability of a polynomial of this format being invertible modulo q high according to [26]. If $N = 2M + 1$ also is a safe prime[6] the probability that our random private key f is invertible is at least

$$1 - \frac{2}{2^M} = 1 - \frac{1}{2^{M-1}}.$$

Since $f(1) = 1$ and $f * f_q = 1 \pmod q$ we have $f_q(1) = 1 + kq$, for some $k \in \mathbb{Z}$. By construction $g(1) = 1$. Thus

[6] A safe prime N is a prime such that $M = (N-1)/2$ also is a prime. The smaller prime is called a Sophie Germain prime, named after the french mathematician with the same name. Generally safe primes are very useful in cryptography. For example in RSA the two private primes typically are safe primes.

$$\sum_{i=0}^{N-1} h_i = h(1) = g(1)f_q(1) = 1 + kq.$$

Now consider the vector

$$(1, 1, \ldots, 1, -k, -k, \ldots, -k) = (\mathbf{1}, -\mathbf{k}),$$

where $\mathbf{1}$ and $\mathbf{k}$ refer to vectors of length N with all elements being equal to 1 and k respectively. If we multiply this vector with the NTRU lattice basis we get

$$\begin{pmatrix} \mathbf{1} & -\mathbf{k} \end{pmatrix} \begin{pmatrix} I & H \\ 0 & qI \end{pmatrix} = \begin{pmatrix} \mathbf{1} & h(1) \cdot \mathbf{1} - qk\mathbf{1} \end{pmatrix} = \begin{pmatrix} \mathbf{1} & \mathbf{1} + qk\mathbf{1} - qk\mathbf{1} \end{pmatrix} = \begin{pmatrix} \mathbf{1} & \mathbf{1} \end{pmatrix}.$$

Thus the vector with all elements equal to 1 is always a vector in L using this format of h. Obviously this vector and any vector parallel to it are not valid solutions to the NTRU Challenge.

Chapter 4
BKZ

Much of the descriptions in this chapter is based on the PhD thesis [3], which also is a reference for a more thorough explanation of the lattice reduction algorithms covered in this chapter. Since the focus of the thesis is on the newest improvements of BKZ, rather than the basic algorithm, the explanations in this chapter are kept quite compact.

4.1 Gram-Schmidt Orthogonalization Process

Before we can examine lattice reduction algorithms in detail we need the Gram-Schmidt orthogonalization process.

Definition 4.1. *(Gram-Schmidt Orthogonalization Process). Consider a set of linearly independent vectors* $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$ *in* $\mathbb{R}^n$*. The Gram-Schmidt Orthogonalization (GSO) family is the set of vectors* $\{\mathbf{b}_1^*, \mathbf{b}_2^*, \ldots, \mathbf{b}_n^*\}$ *in* $\mathbb{R}^n$ *recursively defined as*

$$\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*,$$

where for all $1 \leq j < i \leq n$

$$\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{||\mathbf{b}_j^*||^2}.$$

By construction all the vectors in the GSO are orthogonal towards each other. In principle this would be an excellent basis to use for a lattice, except for the fact that since the coefficients $\mu_{i,j}$ mostly are not integers, the basis vectors are not vectors in the lattice and can not be used as a basis. The GSO process is still an important part of lattice reduction, as explained in Section 4.2.

4.2 LLL

In 1982 Lenstra, Lenstra and Lovász developed the Lenstra-Lenstra-Lovász (LLL) lattice reduction algoritm to be able to factorize polynomials with integer coeffi-

cients into irreducible factors in polynomial time [16]. In the context of this report LLL is used to reduce the basis of lattices. While being fast LLL does not, sufficiently in itself, reduce a basis for our purposes. It is however used as a subroutine in slower but better reduction algorithms.

Before explaining the LLL algorithm we need some notions of what it means for a basis to be reduced. The easiest to achieve, but also weakest notion of lattice reduction, is simply called size reduction.

Definition 4.2. *(Size-reduced). A basis* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$ *of a lattice* $L \subset \mathbb{R}^n$ *is called size-reduced if the GSO-coefficients* μ_{ij} *satisfy* $|\mu_{ij}| \leq 1/2$*, for all* $1 \leq j < i \leq n$*.*

Pseudo-code for size reducing a basis is contained in Algorithm 4.1, where $\lceil \cdot \rfloor$ denotes rounding of a number to the nearest integer.

```
Input  : A basis B = {b_1, b_2, ..., b_n} of a lattice L
Output: A size reduced basis of B = {b_1, b_2, ..., b_n} of L
Calculate the GSO coefficients μ_ij;
for i = 2 to n do
    for j = i − 1 downto 1 do
        b_i ← b_i − ⌈μ_ij⌋ b_j;
        for k = 1 to j do
            μ_ik ← μ_ik − ⌈μ_ij⌋;
        end
    end
end
```

Algorithm 4.1: Size Reduction Algorithm

The size reduction algorithm essentially is the GSO process, but with the coefficients rounded to integers. The next notion of reduction is LLL reduction.

Definition 4.3. *(LLL-reduced). A basis* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$ *of a lattice* $L \subset \mathbb{R}^n$ *is called LLL-*ϵ *reduced with a factor* ϵ*,* $0 < \epsilon < 3/4$*, if it is size-reduced and*

$$||\mathbf{b}^*_{i+1} + \mu_{i+1,i}\mathbf{b}^*_i||^2 \geq (1-\epsilon)||\mathbf{b}^*_i||^2,$$

for $1 \leq i < n$*. The second condition is called the Lovász condition. If a certain value of* ϵ *is implied (typically* $\epsilon = .01$*) the basis is simply called LLL-reduced.*

A way of achieving an LLL-reduced basis is explained in Algorithm 4.2. The algorithm essentially performs size-reduction and swaps basis vectors if it finds an index where the Lovász condition is not satisfied.

4.3 BKZ

The Block Korkine Zolotarev (BKZ) lattice reduction algorithm is originally from [20]. It generalizes the LLL algorithm by introducing a blocksize $\beta \geq 2$. To in-

Input : A basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$ of a lattice L, a parameter ϵ
Output: An LLL-ϵ reduced basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$ of L

$i = 2$;
while $i \leq n$ **do**
  Size-reduce (B_i);
  if $||\mathbf{b}^*_{i+1} + \mu_{i+1,i}\mathbf{b}^*_i||^2 \geq (1-\epsilon)||\mathbf{b}^*_i||^2$ **then**
    Swap$(\mathbf{b}_i, \mathbf{b}_{i-1})$;
    if $i > 2$ **then**
      $i \leftarrow i - 1$;
    end
  else
    $i \leftarrow i + 1$;
  end
end

Algorithm 4.2: LLL Reduction Algorithm

troduce the algorithm we first need to define what it means for a basis to be BKZ reduced.

Definition 4.4. *(BKZ-reduced). A basis* $\{\mathbf{b_1}, \mathbf{b_2}, \ldots, \mathbf{b_n}\}$ *of a lattice* $L \subset \mathbb{R}^n$ *is called BKZ-*β *reduced with a blocksize* $\beta \geq 2$*, if it is LLL-reduced and for each* $1 \leq j < n$*,* $\mathbf{b}^*_j = \lambda_1(L_{[j,k]})$*, where* $k = min(j + \beta - 1, n)$*.*

In other words LLL is a special case of BKZ with $\beta = 2$. The description of the algorithm is explained on three different levels, first the outermost level, then the each so called tour and finally the so called enumeration is explained. Pseudo-code for the outermost level is found in Algorithm 4.3.

Input : A basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$, a blocksize $\beta \in \{2, \ldots, n\}$
Output: A BKZ-β reduced basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$

$\mathbf{B} \leftarrow$ LLL$(\mathbf{B})$; `/* LLL-reduce the basis B */`
Calculate the GSO coefficients U and GSO vectors $\mathbf{B}^*$;
$flag \leftarrow true$;
while $flag = true$ **do**
  $(flag, \mathbf{B}) \leftarrow$ BKZTour$(\mathbf{B}, \beta$,U$, \mathbf{B}^*)$;
end

Algorithm 4.3: BKZ Reduction Algorithm

On this outermost level the algorithm essentially consists of first LLL-reducing the basis and then running BKZ tours on it until such a tour does not change the basis, or in other words returns the value *false*. Pseudo-code for each such tour is found in Algorithm 4.4.

Input : A basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$, a blocksize $\beta \in \{2, \ldots, n\}$, the GSO coefficients U and the GSO vectors $\mathbf{B}^*$

Output: The basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$ after one tour of BKZ, $flag$ with the value *true* if the tour changed the basis and $false$ otherwise

1 $j \leftarrow 0$;
2 $flag \leftarrow false$;
3 **for** $j = 1$ **to** $n - 1$ **do**
4 /* j defines the beginning of the local block */
$k \leftarrow \min(j + \beta - 1, n)$; /* Defining k in local block */
5 $h \leftarrow \min(k + 1, n)$; /* Defining h in local block */
6 $\mathbf{v} \leftarrow \text{Enum}(L_{[j,k]})$; /* Find $\mathbf{v} = \{v_j, \ldots, v_k\} \in \mathbb{Z}^{k-j+1} \setminus \{\mathbf{0}\}$ such that $||\pi_j(\sum_{i=j}^{k} v_i \mathbf{b}_i)|| < ||\mathbf{b}_j^*||$ */
7 **if** $\mathbf{v} \neq (1, 0, \ldots, 0)$ **then**
8 $flag = true$; /* Enumeration was successful */
9 $\text{LLL}(\mathbf{b}_1, \ldots, \mathbf{b}_{j-1}, \sum_{i=j}^{k} v_i \mathbf{b}_i, \mathbf{b}_j, \ldots, \mathbf{b}_h)$; /* Insert $\mathbf{v}$ before the current block, then do LLL-reduction */
10 **else**
11 $\text{LLL}(\mathbf{b}_1, \ldots, \mathbf{b}_h)$; /* LLL-reduce the next block before the next enumeration */
12 **end**
13 Update the GSO coefficients U and GSO vectors $\mathbf{B}^*$;
14 **end**

Algorithm 4.4: BKZ Tour Algorithm

The parameter j defines the beginning of the local block. This parameter sweeps through the interval $\{1, \cdots, n\}$. The parameter k defines the end of this block which has the blocksize β except for the last indices (when $j > n - \beta$ the end of the block k is still n which shrinks the blocksize to $n - j + 1$). The enumeration algorithm searches through the projected sublattice $L_{[j,k]}$ for a vector $\mathbf{v}$ with size less than $||\mathbf{b}_j^*||$. If such a $\mathbf{v}$ is found $\mathbf{v}$ is added to the basis. Finally part of the basis is LLL-reduced to make sure that the basis as a whole basis still is LLL-reduced. If a vector was added to the basis the last vector of the LLL reduced part is removed, to make the basis have the correct number of vectors.

If any new vector was inserted into the basis the $flag$ returns the value *true* and otherwise the value $false$. If no vector was added to the basis this means that $\mathbf{b}_j^* = \lambda_1(L_{[j,k]})$, for all $j \in \{1, \ldots, n\}$. In other words that the basis is BKZ-β reduced. The enumeration algorithm is explained in Section 4.4.

4.4 Enumeration

In this context of explaining the BKZ algorithm the enumeration is a subroutine for finding short vectors in a local block. However, the same algorithm can of course be applied to the entire basis for finding a short vector in the entire basis. Typically a short vector in the entire basis is found by first doing a lattice reduction on the entire basis and then enumerating through the entire reduced basis. In the lattice reduction enumeration is performed on the local basis.

Enumerating through a basis $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$ essentially consists of trying all possible vectors $\mathbf{v} = \sum_{i=1}^{n} x_i \mathbf{b}_i$, where $x_i \in \mathbb{Z}$, looking for a vector with a norm less than a certain enumeration radius R. This set of vectors is infinite so it has to be limited somehow. The following explanation of why there are only a finite number of possible vectors to try is from [28]. From the GSO we have that

$$\mathbf{b}_i = \mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*.$$

Thus a lattice vector can be written as

$$\mathbf{v} = \sum_{i=1}^{n} x_i \mathbf{b}_i = \sum_{i=1}^{n} x_i \left(\mathbf{b}_i^* + \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^* \right) = \sum_{j=1}^{n} \left(x_j + \sum_{i=j+1}^{n} x_i \mu_{i,j} \right) \mathbf{b}_j^*.$$

Projections of $\mathbf{v}$ can now be written as

$$\pi_k(\mathbf{v}) = \pi_k \left(\sum_{j=1}^{n} \left(x_j + \sum_{i=j+1}^{n} x_i \mu_{i,j} \right) \mathbf{b}_j^* \right) = \sum_{j=k}^{n} \left(x_j + \sum_{i=j+1}^{n} x_i \mu_{i,j} \right) \mathbf{b}_j^*.$$

Since the GSO vectors are orthogonal towards each other by construction we have

$$\|\pi_k(\mathbf{v})\|^2 = \left\| \sum_{j=k}^{n} \left(x_j + \sum_{i=j+1}^{n} x_i \mu_{i,j} \right) \mathbf{b}_j^* \right\|^2 = \sum_{j=k}^{n} \left(x_j + \sum_{i=j+1}^{n} x_i \mu_{i,j} \right)^2 \|\mathbf{b}_j^*\|^2. \quad (4.1)$$

We have the following inequalities

$$\|\pi_n(\mathbf{v})\|^2 \leq \|\pi_{n-1}(\mathbf{v})\|^2 \leq \ldots \leq \|\pi_1(\mathbf{v})\|^2 \leq R^2. \tag{4.2}$$

By combining (4.1) and (4.2) we get the following n inequalities

$$\sum_{j=k}^{n} \left(x_j + \sum_{i=j+1}^{n} x_i \mu_{i,j} \right)^2 \|\mathbf{b}_j^*\|^2 \leq R^2, \tag{4.3}$$

for $k \in \{1, \ldots, n\}$. For $k = n$ we get

$$x_n^2 \leq R^2 / \|\mathbf{b}_n^*\|^2 \Leftrightarrow -R/\|\mathbf{b}_n^*\| \leq x_n \leq R/\|\mathbf{b}_n^*\|.$$

Thus x_n can only take a finite number of values. Next rewrite (4.3) as

$$\left(x_k + \sum_{i=k+1}^{n} x_i \mu_{i,k} \right)^2 \leq \frac{R^2 - \sum_{j=k+1}^{n} \left(x_j + \sum_{i=j+1}^{n} x_i \mu_{i,j} \right)^2 \|\mathbf{b}_j^*\|^2}{\|\mathbf{b}_k^*\|^2}.$$

Now fix values $x_{k+1} = x_{k+1}^{'}, \ldots, x_n = x_n^{'}$. Then we can limit x_k to the interval

$$-\sum_{i=k+1}^{n} x_i^{'} \mu_{i,k} - K \leq x_k \leq -\sum_{i=k+1}^{n} x_i^{'} \mu_{i,k} + K,$$

where

$$K = \frac{\sqrt{R^2 - \sum_{j=k+1}^{n} \left(x_j^{'} + \sum_{i=j+1}^{n} x_i^{'} \mu_{i,j} \right)^2 \|\mathbf{b}_j^*\|^2}}{\|\mathbf{b}_k^*\|}.$$

By induction we can thus limit all integers $\{x_1, \ldots, x_n\}$ to finite intervals. The enumeration algorithm essentially tests all of these possible combinations of $\{x_1, \ldots, x_n\}$. If it finds a vector $\mathbf{v} = \sum_{i=1}^{n} x_i \mathbf{b}_i$ with norm less than R it returns that vector and if no such vector exists it returns *failure*. Pseudo-code for the basic enumeration algorithm can be found in Algorithm 4.5.

The enumeration works with a tree with n levels. At level $k \in \{1, \ldots, n\}$ the algorithm tries to vary the coefficient x_k such that (4.3) is satisfied, where the left hand side of the inequality is denoted ρ_k. When (4.3) is satisfied the algorithm goes down to level $k-1$ and calculates ρ_{k-1} as

$$\rho_{k-1} = \rho_k + \left(x_k + \sum_{i=k}^{n} x_i \mu_{i,k-1} \right)^2 \|\mathbf{b}_k^*\|^2. \tag{4.4}$$

By construction $\{\rho_n, \ldots, \rho_1\}$ is an increasing sequence. To minimize ρ_{k-1} the algorithm first tries $x_{k-1} = \lceil -\sum_{i=k}^{n} x_i \mu_{i,k-1} \rfloor = x_{k-1}^{'}$. Then it tries

$$x_{k-1}^{'} + 1, x_{k-1}^{'} - 1, x_{k-1}^{'} + 2, x_{k-1}^{'} - 2, \ldots \text{ until } \rho_{k-1} \geq R^2.$$

To cut the work in half and guarantee that the algorithm does no test both $\mathbf{v}$ and $-\mathbf{v}$ the last non-zero coefficient always has a positive value.

Input : A basis $B = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n\}$, an enumeration radius R.
Output: A vector $v \in L(\mathbf{B})$ such that $||v|| < R$ or *failure* if no such vector exists in $L(\mathbf{B})$.

Calculate the GSO coefficients $\mu_{i,j}$ and GSO vectors $\mathbf{b}_k^*$;
$(x_1, \ldots, x_n) \leftarrow (1, 0, \ldots, 0)$; /* Current combination */
$(\rho_1, \ldots, \rho_n, \rho_{n+1}) \leftarrow (0, 0, \ldots, 0)$; /* $\rho_{n+1} = 0$ */
$(c_1, \ldots, c_n) \leftarrow (0, 0, \ldots, 0)$; /* $c_k = \sum_{j=1}^{n} x_j \mu_{j,k-1}$ */
$(w_1, \ldots, w_n) \leftarrow (0, 0, \ldots, 0)$; /* Jumps frem previous c_k */
$k \leftarrow 1, last\_nonzero \leftarrow 1$; /* Last k such that $x_k \neq 0$ */
while *true* **do**
  $\rho_k \leftarrow \rho_{k+1} + (x_k + c_k)^2 \cdot ||\mathbf{b}_k^*||^2$;
  if $\rho_k < R$ **then**
    if $k = 1$ **then**
      return $\sum_{j=1}^{n} x_j \mathbf{b}_j$;
    else
      $k \leftarrow k - 1$;
      $c_k \leftarrow \sum_{j=1}^{n} x_j \mu_{j,k-1}$;
      $x_k \leftarrow \lceil -c_k \rfloor$;
      $w_k \leftarrow 1$;
    end
  else
    $k \leftarrow k + 1$;
    if $k = n + 1$ **then**
      return *failure*;
    end
    if $k \geq last\_nonzero$ **then**
      $last\_nonzero \leftarrow k$;
      $x_k \leftarrow x_{k+1} + 1$; /* Only enumerate positive half */
    else
      if $x_k > c_k$ **then**
        $x_k \leftarrow x_k - w_k$;
      else
        $x_k \leftarrow x_k + w_k$;
      end
      $w_k \leftarrow w_k + 1$;
    end
  end
end

Algorithm 4.5: Enumeration Algorithm

If the algorithm finds a vector that satisfies $\rho_1 < R^2$, that vector is returned. If the algorithm reaches $k = n + 1$ there exists no short enough vector in the lattice and the algorithm returns *failure*.

4.5 Pruned Enumeration

If ρ_k is close to, but still below R^2 for a big integer k even though (4.3) is satisfied it is very unlikely that that branch contains a short enough vector. The idea of pruned enumeration is to discard that branch.

More precisely pruned enumeration uses a sequence $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$ satisfying $R_1^2 \leq R_2^2 \leq \cdots \leq R_n^2 = R^2$. This sequence is called a bounding function. Instead of demanding $\rho_k < R^2$ the algorithm demands $\rho_k < R_{n-k+1}^2$. In other words line 9 in Algorithm 4.5 is changed to $\rho_k < R_{n-k+1}^2$.

Pruned enumeration was first introduced by Schnorr and Euchner in [22]. Their bounding functions were chosen such that the risk of missing a solution was minimal, while speeding up the algorithm significantly. We will revisit pruned enumeration in Chapter 5, where we discuss extreme pruning.

4.6 Example

A complete example of different lattice reduction algorithms applied on a toy NTRU example lattice can be found in Appendix A.

Chapter 5
BKZ 2.0

For many years the 1997 Number Theory Library (NTL) implementation of BKZ, with Schnorr-Euchner pruned enumeration, was the best BKZ algorithm available at [25]. The first major improvement was made by Chen and Nguyen in 2012 [4]. The improvements mostly covered the enumeration subroutine, which was a bottleneck for BKZ in higher dimensions. The four biggest improvements are discussed in Sections 5.1-5.4 below. Given that the next improvement of the BKZ algorithm, the progressive BKZ, described in Chapter 6, is the focus of this thesis, the descriptions will be quite concise.

5.1 Early Abort

The most obvious improvement is aborting the algorithm prematurely. As is shown in [10], it is possible to make an exponential speed-up by aborting the algorithm early, without the quality of the reduced basis having to suffer too much. They do not discuss this in much detail in the BKZ 2.0 paper and since BKZ 2.0 is not the focus of this thesis, it is enough to know the idea of aborting early.

5.2 Optimized Pruned Enumeration

In their groundbreaking paper [8] Gamma, Nguyen and Regev made two improvements to the pruned enumeration algorithm.

5.2.1 More Precise Pruned Enumeration

First of all they developed a method for picking the Bounding function $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$ from Section 4.5, such that the probability of finding a vector shorter than the enumeration radius R is p, for any $p \in (0, 1]$. This value p is called the pruning probability.

To simplify the analysis some simplifying assumptions are made. First of all it is assumed that the lattice L we enumerate has a unique shortest vector $\mathbf{v} = \lambda_1(L)$ (up to sign) and that we have a good estimation R of that vector to use as the enumeration radius, such that the only vector in L with norm less than R is $\mathbf{v}$.

Some heuristic assumptions are also made. First of all the GH is made, that is

Heuristic 1. *Given a lattice L and a nice set S the number of points in $L \cap S$ is approximately $vol(S)/vol(L)$.*

For the bounding function $\mathbf{R}$, let us define the k-dimensional cylinder-intersection as the set

$$C_{R_1,\ldots,R_k} = \left\{ (x_1,\ldots,x_k) \in \mathbb{R}^k : \sum_{l=1}^{j} x_l^2 \leq R_j^2, \forall j \leq k \right\},$$

for $1 \leq k \leq n$. The set at level k in the pruned enumeration tree is the set of points in the projected lattice $\pi_{n-k+1}(L)$ that are inside $C_{R_1,\ldots,R_k}$. Thus according to Heuristic 1 the estimated number of points in the enumeration tree is approximately

$$N = N_{R_1,\ldots,R_n}(\|\mathbf{b}_1^*\|,\ldots,\|\mathbf{b}_n^*\|) = \frac{1}{2}\sum_{k=1}^{n} \frac{V_{R_1,\ldots,R_n}}{\prod_{i=n-k+1}^{n}\|\mathbf{b}_i^*\|}, \tag{5.1}$$

where $V_{R_1,\ldots,R_n}$ denotes the volume of $C_{R_1,\ldots,R_k}$ and the fact that the volume of $\pi_{n-k+1}(L)$ is $\prod_{i=n-k+1}^{n}\|\mathbf{b}_i^*\|$ is used. Here they calculated $V_{R_1,\ldots,R_n}$ using Monte-Carlo methods. To speak of probabilistic results of a deterministic algorithm like pruned enumeration we must make some assumption on the the distribution of $\mathbf{v}$. We state one more heuristic needed to estimate the p as a function of the bounding function $\mathbf{R}$.

Heuristic 2. *The distribution of $\mathbf{v}$ when written in the normalized GSO basis $\{\mathbf{b}_1^*/\|\mathbf{b}_1^*\|,\ldots,\mathbf{b}_n^*/\|\mathbf{b}_n^*\|\}$ of the input basis look like that of a uniformly distributed vector of norm $\|\mathbf{v}\|$.*

This heuristic was motivated empirically by the authors. Now assume that $\mathbf{v}$ has the coefficients $(x_1,\ldots,x_n)$ in the basis$\{\mathbf{b}_n/\|\mathbf{b}_n\|,\ldots,\mathbf{b}_1/\|\mathbf{b}_1\|\}$ (notice the reversed order of the coordinates). By definition $\mathbf{v}$ belongs to the pruned enumeration tree if and only $\sum_{j=1}^{n} x_j^2 \leq R_k^2$, for all $k \in (1,\ldots,n)$. According to Heuristic 2 $\mathbf{v}$ is distributed like a uniform vector with the extra constraint $\|\mathbf{x}\| = \|\mathbf{v}\|$. The probability that $\mathbf{v}$ is in the pruned enumeration tree can thus be written

$$\Pr_{\mathbf{u}\sim S^{n-1}\|\mathbf{v}\|/R}\left(\sum_{l=1}^{j} u_l^2 \leq \frac{R_j^2}{R_n^2} : \forall j \in \{1,\ldots,n\}\right), \tag{5.2}$$

where $\mathbf{u} \sim S^{n-1}\|\mathbf{v}\|/R$ refers to uniformly picking a vector $\mathbf{u}$ from a sphere in $\mathbb{R}^n$ with radius $\|\mathbf{v}\|/R$. They also discussed how to numerically pick the bounding function $\mathbf{R}$, such that the number of nodes in the pruned enumeration tree (5.1) is minimized, subject to that the pruning probability in (5.2) is a fixed probability p.

5.2.2 Extreme Pruning

They also noticed that an enumeration with a small pruning probability p is significantly faster than $1/p$ times the speed of the enumeration with $p = 1$. Prior to

this it was thought that enumeration should be done with p close to 1. By instead doing many enumerations with a much lower p the enumeration process can be finished much faster while keeping the probability that at least one enumeration is successful close to 1[1].

Since the algorithm is entirely deterministic, to be able to do multiple enumerations with different results the basis has to be randomized between each enumeration. This is done by multiplying the basis with a random unimodular matrix. How a random unimodular matrix can be created is explained in Subsection 6.6.1.

5.3 Preprocessing of the Local Basis

In the enumeration of the local block in the BKZ algorithm the local block is not guaranteed to be better reduced than LLL. In BKZ 2.0 the local block is preprocessed with a BKZ reduction with a smaller blocksize α first.

This inner BKZ reduction can in turn use preprocessing of the local blocks with an even smaller blocksize. If this recursive strategy is used at some point there has to be a base case blocksize where no deeper inner preprocessing is used.

5.4 Optimizing the Enumeration Radius

Empirically it turns out that, except for the last indices, the norm of the final vector in the enumeration of a local block $L_{[j,k]}$ is mostly between 0.95 and 1.05 times the GH of the local block, whereas the ratio between the norm of the first vector and the GH is mostly almost 1.1. To lower the enumeration radius and thus speed up the enumeration they use the enumeration radius
$R = min(\sqrt{1.1}GH(L_{[j,k]}), ||\mathbf{b}_j^*||)$, except for the last 30 indices where they use $R = ||\mathbf{b}_j^*||$.

[1] If, for example, N enumerations are done with a pruning probability of $1/N$ the probability of at least one enumeration succeeding is $\lim_{N\to\infty} 1-(1-1/N)^N =$
$= \lim_{N\to-\infty} 1-(1+1/N)^{-N} = 1-1/e$. The pruning probability can of course be increased if a bigger total success probability is wanted.

Chapter 6
Progressive BKZ

6.1 Introduction

In May this year Aono et al. published a new BKZ algorithm and also some new ideas for lattice enumeration [2]. While not being as groundbreaking as the paper on extreme pruning, it does contain many interesting optimizations. We will go though the major improvements, starting from the innermost optimizations of pruning probability and enumeration radius and gradually working ourselves out to how to compromise between BKZ reduction and vector enumeration.

First introduce some notation that is needed in this chapter.

6.1.1 Notation

Let $[i:j]$, where $j > i$, denote the set $\{i, \ldots, j\}$ and $[j]$ denote the set $[1:j]$. For convenience, in this chapter we let B_i denote the lattice $L_{[i,j]}$, if the ending index j is clear by the context. For a blocksize β and an index i we use the notation $\beta' = \min(i + \beta - 1, n)$ for the size of the local block B_i. Given the basis B of a lattice L, we introduce the expression

$$\mathrm{FEC}(B) = \sum_{k=1}^{n} \frac{V_k(\mathrm{GH}(L))}{\prod_{i=n-k+1}^{n} \|\mathbf{b}_i^*\|}. \tag{6.1}$$

The Full Enumeration Cost (FEC) is the expected number of processed nodes in the enumeration tree with enumeration radius $\mathrm{GH}(L)$ (the expected value of $\lambda_1(L)$ given that the GH is true). This is in other words a measure of the expected computational work of enumerating the lattice to find the shortest vector. Given a bounding function $\mathbf{R} = \{R_1^2, \ldots, R_n^2\}$, just as in Subsection 5.2.1, the expected number of points in the pruned enumeration tree with enumeration radius R is

$$N = N_{R_1,\ldots,R_n}(\|\mathbf{b}_1^*\|, \ldots, \|\mathbf{b}_n^*\|) = \frac{1}{2} \sum_{k=1}^{n} \frac{V_{R_1,\ldots,R_n}}{\prod_{i=n-k+1}^{n} \|\mathbf{b}_i^*\|}. \tag{6.2}$$

The pruning probability is

$$\Pr_{\mathbf{u}\sim R\cdot S^{n-1}}\left(\sum_{l=1}^{j} u_l^2 \leq R_j^2 : \forall j \in \{1,\ldots,n\}\right). \tag{6.3}$$

We denote the minimum value of (6.2), given that (6.3) is fixed to $p \in (0,1]$ and the enumeration radius is $R = \alpha \cdot \mathrm{GH}(L)$, by $\mathrm{ENUMCost}(B;\alpha,p)$. We use the analogous definition for a projective sublattice B_i. In the progressive BKZ paper some heuristic assumptions are made.

6.1.2 Heuristic Assumptions

In [21] Schnorr introduced the Geometric Series Assumption (GSA), which says that the squared norms of the GSO vectors decay geometrically in a BKZ-reduced basis. More specifically the GSA states that $\|\mathbf{b}_i^*\|^2/\|\mathbf{b}_1\|^2 = r^{i-1}$, for some $r \in [3/4,1)$. Generally the GSA approximately holds except for the last and the first indices, where generally $\|\mathbf{b}_i^*\|$ is lower than the GSA suggests.

In [4, Full version, appendix C] Chen and Nguyen noticed that the norm of the shortest vector in a local block B_i usually is larger than $\mathrm{GH}(B_i)$, for small dimensions β'. For $1 \leq i \leq 50$ define the modified GH constant as

$$\tau_i = \frac{\lambda_1(\pi_{n-i+1}(L))}{\mathrm{GH}(\pi_{n-i+1}(L))}. \tag{6.4}$$

These constants were estimated empirically by Chen and Nguyen. Later in this chapter in Section 6.3 it will be assumed that

$$\lambda_1(B_i) \approx \begin{cases} \tau_{\beta'} \cdot \mathrm{GH}(B_i), & \text{if } \beta' \leq 50 \\ \mathrm{GH}(B_i), & \text{if } \beta' > 50 \end{cases}. \tag{6.5}$$

6.2 Optimizing Plain BKZ

The paper starts off by optimizing what they call plain BKZ, explained by the pseudo-code in Algorithm 6.1. The optimization here consists of picking α and p as a function of β, n and i.

6.2.1 Basic Parameter Settings

Under the GSA the authors showed numerically that the optimal values of α, p and r satisfy the following three equations

$$p = \frac{2}{\alpha^\beta}, \tag{6.6}$$

$$r = \left(\frac{\beta+1}{\alpha\beta}\right)^{\frac{4}{\beta-1}} \cdot V_\beta(1)^{\frac{4}{\beta(\beta-1)}}, \tag{6.7}$$

Input : A basis B of dimension n, a blocksize β
Output: A BKZ-β reduced basis B

$B \leftarrow \text{LLL}(B)$;
$flag \leftarrow true$; /* $flag = true$ if the basis is updated */
while $flag = true$ **do**
  $flag \leftarrow false$;
  for $i = 1$ **to** $n - 1$ **do**
    Set (α, p) for local block B_i of blocksize $\beta_i' = \min(\beta, n - i + 1)$;
    Execute lattice enumeration with probability p
    and radius $\alpha \cdot \text{GH}(B_i)$;
    Save result in vector $\mathbf{v}$;
    if $||\mathbf{v}|| < \alpha \cdot GH(B_i)$ **then**
      Update basis B by $\mathbf{v}$;
      $flag \leftarrow true$;
    end
  end
end

Algorithm 6.1: The Plain BKZ Algorithm

$$\log(r) = \begin{cases} -18.2139/(\beta + 318.978), & \text{if } \beta \leq 100 \\ (-1.06889/(\beta - 31.0345)) \cdot \log(0.417419\beta - 25.4889), & \text{if } \beta > 100 \end{cases}, \tag{6.8}$$

for a blocsize β. Solving these equations gives us the so called basic parameter setting. The authors also define another function which measures the minimum cost of enumeration with blocksize β, called MINCost(β). Numerically they estimate it as

$$\log_2(\text{MINCost}(\beta)) = \begin{cases} 0.1375\beta + 7.153, & \text{if } \beta \in [60, 105] \\ 0.000898\beta^2 + 0.270\beta - 16.97, & \text{if } \beta > 105 \end{cases}. \tag{6.9}$$

It is not clear from their article what value MINCost(β) takes for $\beta < 60$. In Subsection 6.2.2 we will cover how to modify these settings for the first and last indices where the GSA no longer holds.

6.2.2 Modified Parameter Settings

Having calculated p and α for a blocksize β and an index i the following modifications are made.

Modifications for the First Indices

For small indices i, (it is not clear how small from their article), the blocksize of the local block is increased to the smallest blocksize β such that $\text{ENUMCost}(B_{i:i+\beta-1}; \alpha, p) > \beta \cdot \text{MINCost}(\beta)$.

Modifications for the Last Indices

For indices $i > n - \beta$ the blocksize of the local block shrinks to $\beta' = n - i + 1$. Then the pruning probability p' is increased until $\text{ENUMCost}(B_i, \alpha', p')$ equals $\beta \cdot \text{MINCost}(\beta)$, where $\alpha' = (2/p')^{1/\beta}$ from (6.6).

6.3 Basic Progressive BKZ

Next, the paper introduces the basic variant of the progressive BKZ algorithm, as explained in Algorithm 6.2.

Input : A basis B of dimension n, a starting blocksize β_{start}, an ending blocksize β_{end}
Output: A reduced basis B

$B \leftarrow \text{LLL}(B)$;
for $\beta = \beta_{start}$ **to** β_{end} **do**
  while *FEC(B)>Sim-FEC(n, β)* **do**
    for $i = 1$ **to** $n - 1$ **do**
      Set (α, p) for local block B_i
      of blocksize $\beta_i' = \min(\beta, n - i + 1)$;
      Preprocess B_i by the progressive BKZ;
      Execute lattice enumeration with probability p
      and radius $\alpha \cdot \text{GH}(B_i)$;
      Save result in vector $\mathbf{v}$;
      if $||\mathbf{v}|| < \alpha \cdot GH(B_i)$ **then**
        Update basis B by $\mathbf{v}$;
      end
    end
  end
end

Algorithm 6.2: Basic Progressive BKZ Algorithm

The idea of progressive BKZ is to start off doing BKZ-tours (each while loop in Algorithm 6.2 is called a tour) with a small blocksize $\beta = \beta_{start}$ and then increase the blocksize once a certain criterion is met. Then new tours are done with this new blocksize until the criterion is met again and so on. This keeps on until the criterion is met for the blocksize $\beta = \beta_{end}$. The picking of p and α in each

tour is done the same way as in Algorithm 6.1. As in BKZ 2.0 the local basis is preprocessed before enumeration. Exactly how this is done is not clear from their article.

In basic progressive BKZ the blocksize is increased by one each time. In Section 6.4 we quickly discuss how to pick a better strategy for how to increase the blocksize.

In Algorithm 6.2 the criterion for increasing the blocksize is whether $\text{FEC}(B) > \text{Sim-FEC}(n, \beta)$ or not. Here $\text{FEC}(B)$ is defined in Subsection 6.1.1. This expression decreases as more BKZ tours are performed.

The expression $\text{Sim-FEC}(n, \beta)$ is a simulation of the FEC after having applied Algorithm 6.1 to a basis B with lattice dimension n and blocksize β. It depends only on n and β, in other words, the $\text{Sim-FEC}(n, \beta)$ does not change with the BKZ tours. How $\text{Sim-FEC}(n, \beta)$ is calculated will be discussed in Subsection 6.3.1. First we introduce some more notation.

We denote the lengths of the simulated GSO vectors $(\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*)$ by $(l_1, \ldots, l_n)$ and call them $\text{Sim-GSO-lengths}(n, \beta)$. We also define some functions on the simulated GSO lengths. First define the GH as

$$\text{Sim-GH}(l_1, \ldots, l_n) := \left(\frac{V_n(1)}{\prod_{j=1}^{n} l_j^{1/n}} \right)^{\frac{1}{n}}, \tag{6.10}$$

then define the FEC as

$$\text{Sim-FEC}(l_1, \ldots, l_n) := \sum_{k=1}^{n} \frac{V_k(\text{Sim-GH}(l_1, \ldots, l_n))}{\prod_{i=n-k+1}^{n} l_i}. \tag{6.11}$$

Let us define the $\text{Sim-ENUMCost}(l_1, \ldots, l_n; \alpha, p)$ as $\text{ENUMCost}(B; \alpha, p)$ for a basis B, where $l_i = \|\mathbf{b}_i^*\|$. For $(l_1, \ldots, l_n) = \text{Sim-GSO-lengths}(n, \beta)$ use the notation $\text{Sim-FEC}(n, \beta) := \text{Sim-FEC}(l_1, \ldots, l_n)$.

6.3.1 Simulating the FEC

The simulation of the FEC consists of first simulating the GSO lengths $(l_1, \ldots, l_n)$ and then calculating $\text{Sim-FEC}(l_1, \ldots, l_n)$. Calculating $(l_1, \ldots, l_n)$ consists of two phases.

Phase 1

First let $l_n = 1$ and then work backwards index wise and solve the following equation for l_i

$$l_i = \max\left(\frac{\beta'}{\beta'+1}\alpha, \tau_{\beta'}\right) \cdot \text{GH}(l_i, \ldots, l_{i+\beta'-1}). \tag{6.12}$$

Here α is the optimized parameter from Algorithm 6.1 (it is not entirely clear if the basic or modified α is used) and $\tau_{\beta'}$ is the modified GH constant from Subsection 6.1.2. For $\beta < 30$ the simulated GSO lengths $(l_1, \ldots, l_n)$ from this phase are enough, but if $\beta \geq 30$ phase 2 is applied to $(l_1, \ldots, l_n)$ to change the l_i for small and big i.

Phase 2

Quite analogously with the modification of p and α, l_i are modified for small and big i. For the last indices, that is when $i > n - \beta - 1$, or in other words when $\beta' < \beta$, p_i is changed such that Sim-ENUMCost$(l_1, \ldots, l_n; \alpha_i, p_i) =$ MINCost(β), where $\alpha_i = (2/p_i)^{n-i+1}$. Here the values of $(l_1, \ldots, l_n)$ are taken from phase 1. After having updated (α_i, p_i) for $n - \beta + 1 < i \leq n$ the simulated GSO lengths l_i by solving

$$l_i = \max(\frac{\beta'}{\beta'+1}\alpha_i, \tau_{\beta'}) \cdot \mathrm{GH}(l_i, \ldots, l_n), \tag{6.13}$$

for all i, such that $n - \beta + 1 < i \leq n$, in other words essentially solving (6.12) again.

For the first indices (it is not clear how many indices count as the first) we have an integer $b > 0$ of size enlargement. At index i we reset the blocksize at i as $\beta_i := \beta + \max((b - i + 1)/2, b - 2(i-1))$, where the division by 2 refers to integer division. Using these modified blocksizes the simulated GSO lengths are recomputed by again solving (6.12) from $i = \beta_i$ to 1. Then Sim-ENUMCost$(l_1, \ldots, l_{\beta+b}; \alpha, p)$ is computed. The biggest b, such that the simulation enumeration cost is less than $2 \cdot$ MINCost(β), is picked.

After having finished phase 1 and 2, Sim-FEC$(l_1, \ldots, l_n)$ is calculated by (6.11).

6.4 Optimized Progressive BKZ

We say that a basis B is β-reduced if FEC(B)<Sim-FEC(n,β). For a triple of blocksizes $(\beta^{alg}, \beta^{start}, \beta^{goal})$ satisfying $2 \leq \beta^{start} < \beta^{goal} \leq \beta^{alg}$ the notation

$$\beta^{start} \xrightarrow{\beta^{alg}} \beta^{goal},$$

refers to the process of starting with a β^{start}-reduced basis B and using tours of the BKZ-β^{alg} algorithm, with parameters optimized according to Section 6.2 above, until FEC(B)<Sim-FEC(n,β^{goal}). In other words until the basis is BKZ-β^{goal} reduced. A blocksize strategy is a sequence $\{(\beta_j^{alg}, \beta_j^{goal})\}_{j=1,\ldots,D}$, referring to the following progressive reduction of an LLL-reduced basis B

$$LLL \xrightarrow{\beta_1^{alg}} \beta_1^{goal} \xrightarrow{\beta_2^{alg}} \beta_2^{goal} \xrightarrow{\beta_3^{alg}} \cdots \xrightarrow{\beta_D^{alg}} \beta_D^{goal},$$

where $\beta_D^{goal} = \beta$ is the desired reduction blocksize. One of the most important findings of the progressive BKZ paper is a method for finding the optimal blocksize strategy, to achieve a certain goal blocksize β. Sadly enough it turns out that the optimal blocksize strategy is just marginally better than the simple blocksize strategy, when the blocksize is larger than about 100. Then the simple strategy is better, since it does not need the heavy precomputing of the optimal blocksize strategy. Since we want to use blocksizes above 100 for the NTRU Challenges, we will not discuss the optimal blocksize strategy in further detail.

6.5 Combining Progressive BKZ and Vector Enumeration

Given the task of finding a vector shorter than $\gamma \cdot GH(L)$ in a lattice L with a given basis B the following method is used. First M different randomized bases B_i are generated by multiplying B by M different random unimodular matrices U_i. Each basis is BKZ-β reduced for a certain value β. Finally enumeration with the enumeration radius $\gamma \cdot GH(L)$ and the pruning probability $p = 2 \cdot \gamma^{-n}/M$ is done on each basis. Thus the way progressive BKZ strategy makes use of the extreme pruning strategy from BKZ 2.0, is to apply it to the entire basis, instead of the local block.

Since a random lattice has about $\gamma^n/2$ pairs of vectors smaller than $\gamma \cdot GH(L)$ the probability of success for each enumeration is about $\gamma^n/2 \cdot 2 \cdot \gamma^{-n}/M = 1/M$. The probability of success for at least one enumeration is thus $1-(1-1/M)^M \approx 1-1/e \approx 0.6321$[1]. Obviously, by using a higher pruning probability, the success probability can be increased at the cost of longer computation time.

The optimization problem here is, given a particular lattice, to pick the parameter pair (β, M) that minimizes the computational cost of this process. The optimization assumes that, the optimized and not the simple, blocksize strategy is used. (This could of course modified to better fit the attacks on NTRU given that we want to use the basic blocksize strategy.) However, as is shown in Chapter 7 below, for the NTRU lattice in particular, it is possible to do multiple enumerations on one reduced basis. Also NTRU has a lot more very short vectors than the average random lattice. Thus the analysis behind this method is not really applicable to NTRU lattices and we will not discuss it further.

6.6 Some Other Aspects of Progressive BKZ

6.6.1 How to Randomize a Basis

The algorithm for creating a random unimodular matrix in the implementation of the progressive BKZ algorithm [2], is essentially a slight modification of the SageMath function

[1] $\lim_{M\to\infty} 1-(1-1/M)^M = \lim_{M\to-\infty} 1-(1+1/M)^{-M} = 1-1/e$

random_unimodular_matrix, found in src/sage/matrix/special.py in version 7.1 of SageMath [7]. The default value for the upper element limit value is 1024, but is easily modified. For the purpose of randomizing an NTRU lattice basis, a smaller limit has to be used not to destroy the basis quality.

The algorithm for creating a random unimodular matrix consists of two parts. First a random upper triangular matrix is generated, with the value 1 along the main diagonal. Then for each row in the matrix, a random multiple of each other row is added. These operations do not change the determinant of the matrix and thus the matrix is still unimodular. Each step has to be repeated until the biggest element in the modified row is smaller, in absolute value, than the upper limit.

It should be mentioned, that the generated matrices are by no means uniformly picked among all unimodular matrices with a certain upper limit on the element values. For this application that is not a problem.

6.6.2 Better Method for Finding Optimal Bounding Functions

The method for computing the bounding function for the pruned enumeration is from [1]. This improved method makes the time to compute the bounding function negligible. These computations used to be quite time consuming due to heavy Monte-Carlo computations.

6.6.3 General note on the details of this chapter

It should be mentioned that while the descriptions in this chapter sometimes are very detailed, especially in Subsections 6.2.2 and 6.3.1, there are still more details to understanding the whole progressive BKZ algorithm. Diving too deep into these would make it easy for the reader to loose the overall picture of the algorithm. The interested reader who wants to know more about the progressive BKZ algorithm is referred the original article [2] and especially to its implementation [23].

Chapter 7

BKZ Reduction Attacks Against NTRU Challenges

In this thesis two ways of finding an acceptable vector to the NTRU Challenge are considered.

7.1 BKZ Reduction

The first and most obvious method is to BKZ-β reduce the NTRU lattice with a big enough β, such that the second vector of the reduced basis is a valid solution. The first vector will always be the 1s or -1s only vector and is not a valid solution. This method is the easiest to use but also the slowest.

The simple blocksize strategy was used. It turns out that for a blocksize around $\beta = 100$, the time to precompute the optimal blocksize strategy is so long, that it starts to get faster to just use the simple strategy [2, p. 22-23]. It would be possible to use the optimized strategy up to a certain limit and then switch to the simple strategy, but given that almost all the computational time is spent on the last blocksizes, the potentially saved time would be small.

7.2 BKZ Reduction and BDD Enumeration

The first two NTRU challenges were solved using BKZ reduction only. Challenges 3-7 were solved by Ducas and Nguyen using a modified version of BKZ-reduction followed by enumeration, explained in this section. On Security Innovations web page there is a PDF with explanations of how the challenges were solved [15].

In the NTRU Challenge the private key $(f, g) = (1 + 3F, g)$ and the public key h has the following relationship

$$h = (1 + 3F)^{-1} * g \pmod{q}.$$

This equation can be rewritten as

$$F * 3h = g - h \pmod{q}. \tag{7.1}$$

This modulo expression means that there is a polynomial λ, such that

$$F * 3h + q\lambda = g - h.$$

On matrix-vector form this can be written as

$$\begin{pmatrix} \mathbf{F} & \boldsymbol{\lambda} \end{pmatrix} \begin{pmatrix} I & 3H \\ 0 & qI \end{pmatrix} = \begin{pmatrix} \mathbf{0} & -\mathbf{h} \end{pmatrix} + \begin{pmatrix} \mathbf{F} & \mathbf{g} \end{pmatrix}, \tag{7.2}$$

where the element on row i and column j in H is $h_{ij} = h'_{j-i \pmod{n}}$, if the first row and column are numbered as 0. The block matrix above is referred to as the modified NTRU lattice. To find a vector resembling the private vector, instead of searching for a small vector or in other words searching for a vector close to the zero vector, we search for a vector close to $(\mathbf{0}, -\mathbf{h})$. This vector is called the target vector. The problem of finding a vector close to the target vector is called Bounded Distance Decoding (BDD). BDD is essentially the CVP problem, but with a guarantee that there exist vectors that are much shorter than the GH of the lattice.

To solve BDD, first of all we do a BKZ-reduction on the modified NTRU lattice. The BDD can then be solved by enumeration on the BKZ-reduced modified NTRU lattice, by doing some minor changes in the enumeration algorithm (essentially just searching for a vector close to the target vector instead of close to the zero vector) [17].

Notice that $(\mathbf{F}, \mathbf{g})$ is a much shorter vector than $(\mathbf{f}, \mathbf{g})$. This means that we can use a much smaller enumeration radius making the enumeration much faster. Since we know the private keys of the first seven Challenges we can cheat and calculate the norm of each key $(\mathbf{F}, \mathbf{g})$. We can then increase that value a little and use it as enumeration radius. If we did not know the norm of the private key, we could still generate a number of private keys on the correct format and use these to estimate the norm of the actual private key.

If we multiply (7.1) by $\mathbf{x}^i$ we analogously get the matrix-vector form

$$\begin{pmatrix} \mathbf{F} * \mathbf{x^i} & \boldsymbol{\lambda} \end{pmatrix} \begin{pmatrix} I & 3H \\ 0 & qI \end{pmatrix} = \begin{pmatrix} \mathbf{0} & -\mathbf{h} * \mathbf{x}^i \end{pmatrix} + \begin{pmatrix} \mathbf{F} * \mathbf{x}^i & \mathbf{g} * \mathbf{x^i} \end{pmatrix}. \tag{7.3}$$

Thus a new BDD enumeration can be done on the same basis by just rotating the target vector. This way n different enumerations can be done on the same reduced basis. If we want the probability of finding the solution vector to be 1-ϵ for $\epsilon > 0$, it is enough to have a probability of 1-$\epsilon^{1/n}$ of finding a short vector on each enumeration. If we want an even smaller probability on each enumeration, we can of course use a smaller one and randomize and reduce the original basis again, if all enumerations fail. Then we can do n more enumerations on the new reduced basis and so on. Exactly what pruning probability should be used to achieve a certain probability of success for each enumeration can be estimated empirically.

Chapter 8
Implementation

At the beginning of the project, attempts were made at solving the first NTRU Challenge, using the NTL implementation of BKZ [25], using the first method from Chapter 7. This turned out to be impossible and attempts were made at implementing both the progressive BKZ algorithm and BKZ 2.0. During these attempts the authors behind the progressive BKZ article published their implementation of their algorithm, the progressive BKZ library, available at the National Institute of Information and Communications Technology (NICT) web page [23]. It was then decided that time was better spent doing simulations using their software, than to try to reinvent the wheel and write another implementation of the same algorithm.

The progressive BKZ library contains functions for BKZ reduction and vector enumeration. The original version could only handle SVP enumeration, but after correspondence with the authors CVP enumeration was added in the second version of the library. Also after correspondence with the authors, about enumeration problems when searching for very short vectors in high dimensional lattices, a big bug in the code was found and fixed.

The library is heavily based on NTL. The core of both the BKZ reduction and the vector enumeration is modified from the NTL library. NTL is also used for LLL reduction, representation of big integers and floating point numbers and so on.

The procedure for solving the challenges was the following. First the NTRU lattice or the modified NTRU lattice was generated, using the available public keys from the challenges' web page [13]. Using the first method from Chapter 7, the lattice was then BKZ reduced, using the basic progressive BKZ algorithm, using bigger and bigger blocksize, until the norm of the second basis vector was less than the GH of the basis.

Using the second method from Chapter 7, the modified lattice was BKZ reduced using a big blocksize. Then all the target vectors were created, again by using the public key. The enumerations were then made, searching for vectors close to the target vectors. The pruning probability was empirically tested to be low, so that the enumerations were fast, but big enough, such that at least one enumeration was successful. The norms of the private keys were calculated using the available solutions [15], these norms were then used as lower limits of how low the enumeration radii could be set.

To speed up the enumeration process and make it more automated, a Python script was created that automatically started a new enumeration once the last one finished.

Chapter 9

Results

For convenience, information about the first 7 challenges is summed up in Table 9.1. Challenges 1 and 2 only have solution vectors similar to but not identical to the private vectors available, which explains the lack of norms of their private vectors. Given that Challenges 1 and 2 were solved using BKZ reduction only and that the submitted solutions to Challenges 1 and 2 also only used BKZ reduction, that is not really a problem.

The simulations were done on three computers. Two of the computers had an AMD Phenom(tm) II X4 810 Processor CPU with 4 GB memory, both of them are henceforth referred to as CAS. One of the computers had an Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz with 16 GB memory, henceforth referred to as BLUL. All of the machines had 4 cores. Vector enumeration is very paralellizable and as many available cores as possible were used to do the enumerations. BKZ reduction is not as paralellizable and to be able to run multiple reductions at the same time only one core was used per BKZ reduction.

Table 9.1: The parameter q, the key size n, the GH and the norms of the private keys (where $f = 3F + 1$) corresponding to the first 7 Challenges.

Challenge nr	**q**	**n**	$\lceil \mathbf{GH}(L) \rceil$	$\|\|(\mathbf{f}, \mathbf{g})\|\|$	$\|\|(\mathbf{F}, \mathbf{g})\|\|$
1	512	107	81	-	-
2	1024	113	117	-	-
3	1024	131	126	30.4959	13.4866
4	1024	139	130	31.8119	13.9483
5	1024	149	134	32.4962	14.3720
6	1024	163	140	33.1662	14.7836
7	1024	173	145	33.0151	15.0074

9.1 BKZ Reduction

9.1.1 Challenge 1 and 2

For both the first and the second NTRU Challenge (with $n = 107$ and $n = 113$ respectively), after a couple of days of computation on CAS, the progressive BKZ library managed to do a BKZ-110 reduction on the NTRU lattice. The second vectors of the reduced lattices had the sizes 25.3772 and 28.3549 respectively, which is way below the respective GHs of 81 and 117 (The first vector is always either the 1s or -1s only vector and is thus not allowed as a solution to the challenge).

9.1.2 Challenge 3

After around 10 days of computation on BLUL a BKZ-120 reduction was made on the NTRU lattice for the 3rd Challenge ($n = 131$). The second row of the reduced basis had the size 30.4959, which is way below the GH of 126.

9.1.3 Challenge 4

After first spending several days on doing a BKZ-110 reduction on the 4th Challenge ($n = 139$) on BLUL and then spending another 16 days on BKZ-120 reduce the basis on BLUL, the second shortest vector was still above the GH of 130. This seems to be close to the limit of what is possible to solve with the most basic method, at least using the available hardware.

9.2 BKZ Reduction and BDD Enumeration

9.2.1 Challenge 4

Less than 2 days were spent on BKZ-110 reducing the modified version of the 4th Challenge on BLUL. The private key (F, g) corresponding to the 4th Challenge has a norm of about 13.96. Enumerations using 4 cores on BLUL were made with an enumeration radius of 15 and a pruning probability of 0.01. After less than a day of computation 4 different vectors were found. It was obvious that a too big pruning probability was used. This solution was made even though the block matrix $3H$ was forgotten to be reduced modulo q. The method of BKZ reduction followed by BDD enumeration seemed promising.

9.2.2 Challenge 5

48 hours of computation were made on BLUL to BKZ-110 reduce the modified version of the 5th Challenge ($n = 149$). Then pruned enumerations with radius 15 and pruning probability 0.001 were done on all three computers, using 3 cores on BLUL and all 4 cores on the other 2. After almost 3 days of computation the 52nd enumeration was successful. It was probably still possible to use a lower pruning probability.

9.2.3 Challenge 6

A little more than 30 days were spent on BKZ-120 reducing the sixth Challenge ($n = 163$), on BLUL. Since the time the progressive BKZ library expected each enumeration to take, even using extremely low pruning probability, was too high, hopes of solving the sixth Challenge in this project were abandoned.

Currently (as of 2016-08-17) 7 out of the 27 challenges are solved [13].

9.3 Comparisons to the Submitted Solutions

9.3.1 Project Solutions

Table 9.2 contains the time spent on BKZ reduction, BDD enumeration and in total to solve the NTRU Challenges in this project. The first three challenges were solved using only BKZ reduction and the fourth and fifth were solved using the BKZ and BDD reduction method. The enumeration time was calculated as the total time spent on enumeration, times the number of cores used, divided by the number of solutions found.

For more details on each solved challenge, including how the simulations were distributed between the different cores, see the Sections 9.1-9.2.

9.3.2 First Submitted Solutions

Table 9.3 contains the time spent on BKZ reduction, BDD enumeration and in total for the first submitted solution to each NTRU Challenge.

The first three challenges used the fplll implementation of BKZ reduction [27], which is similar to but somewhat faster than the NTL implementation. Challenges 4-7 used a modified version of the BKZ 2.0 algorithm from [4]. The enumeration used for the last five challenges was a modified version of the BDD algorithm from [17].

The first two challenges were solved by a modified version of the BKZ only method explained in Chapter 7. The last five were solved by the BKZ and BDD enumeration method also covered in Chapter 7.

There is no information about the hardware used for the first two challenges. Challenge 3 used a single core, 3.2 GHz processor. Challenge 4 used a 1.3 GHz processor. Challenge 5 used a 1.3 GHz processor for BKZ reduction and a 2.5 GHz processor for enumeration. Challenges 6 and 7 used a 2.53 GHz processor.

For more details on how each solution was made, the hardware used and so on see [15].

Table 9.2: The time spent on BKZ reduction, BDD enumeration and in total to solve the NTRU Challenges in this project.

Challenge	Proj. BKZ	Proj. Enumeration	Proj. Total
1	< 2 days	-	< 2 days
2	< 2 days	-	< 2 days
3	10 days	-	10 days
4	< 2 days	< 1 day	< 3 days
5	2 days	31 days	33 days

Table 9.3: The time spent on BKZ reduction, BDD enumeration and in total for the first submitted solution to each NTRU Challenge.

Challenge	First BKZ	First Enumeration	First Total
1	10 h	-	10 h
2	8 h	-	8 h
3	12 h	6 h	18 h
4	10 h	1 h	12 h
5	48 h	4 h	52 h
6	30 days	11 days	41 days
7	240 days	50 days	290 days

Chapter 10
Discussion

The progressive BKZ algorithm works well. It was possible to solve the first three NTRU Challenges by BKZ reduction only. The fourth Challenge turned out to be very slow to solve by BKZ reduction only.

It should be said that the first two challenges were solved faster by the fplll BKZ implementation. They did however BKZ reduce a modification of the NTRU lattice, which is probably the explanation why their solution was faster. They did not seem to be able to solve the third challenge, which was possible using the progressive BKZ algorithm, which indicates that progressive BKZ is the faster algorithm, they just used a faster method.

Both the fourth and the fifth NTRU Challenge turned out to be quite easy to solve using the BKZ-reduction plus BDD enumeration strategy.

Then it was not possible to solve more challenges. Given that the challenges increase very fast in computational difficulty it is not that strange that the attack suddenly gets too slow. However, Ducas and Nguyen managed to solve seven challenges. They also managed to solve the fourth and fifth challenge much faster than this project did. There are a number of possible explanations to this.

It is not clear from their solution what enumeration, pruning probability or BKZ blocksize they used. In this project these parameters were picked by trial and error, until a solution was found, and by no means optimized.

According to the progressive BKZ article [2], the progressive BKZ algorithm is around 50 times faster than BKZ 2.0 at solving the TU Darmstadt SVP Challenge [5], up to 160 dimensions. The results in this project do not reflect that fact. The factor 50 however, is based on a lot of assumptions that are not met in this case.

The most interesting challenges, that is Challenge 6 and harder, use lattices with over 300 dimensions. The NTRU lattices have a very special structure, including a lot of extremely short vectors. While both the BKZ reduction and enumeration in the progressive BKZ library are written for very general purposes, the BKZ 2.0 algorithm used by Nguyen and Ducas might very well be optimized for finding short vectors in NTRU lattices. The lattices in the SVP Challenge have the structure of a standard, random lattice, where the shortest vector's norm is about the GH of the lattice.

The structure and size of the SVP Challenge lattices makes it possible to speed up the progressive BKZ significantly, using the optimized blocksize strategy and the optimal mixture of BKZ blocksize and the number of randomized bases to reduce, as is explained in Sections 6.4-6.5. As is also discussed in those sections these tricks do not apply to lattice reduction attacks against NTRU.

The factor 50 is estimated theoretically. It is possible that the implementation of BKZ 2.0 is more efficiently written than the implementation of progressive BKZ. This is not possible to investigate, since the implementation of BKZ 2.0 is not publicly available.

In Section 10.1 some ideas for how to solve more challenges using the progressive BKZ library and how to do further research, are discussed.

10.1 Suggested Further Research

The most obvious possible further research to do is to continue to try the BKZ-reduction plus BDD enumeration strategy on harder NTRU Challenges, using more time and/or more powerful hardware. The lack of these was a bottleneck that made it impossible to solve more challenges. Even solving Challenge 6 took 41 days for Nguyen and Ducas. With significantly more computing power and/or time Challenge 6 should probably be breakable with the progressive BKZ library.

The challenges in this project were solved using quite a lot of trial and error. It would be interesting to investigate how to pick the BKZ blocksize β, the enumeration radius, the pruning probability and the number of randomized bases to BKZ-β reduce M, to make the probability of at least one of the Mn BDD enumerations to succeed, greater than $1 - \epsilon$, for an NTRU Challenge of length n and a parameter $\epsilon > 0$. Next this expression could be optimized to minimize the computation time, subject to the available hardware. After having done this it should be possible to solve the NTRU Challenges faster.

Another idea for further research is to look into the possibility of implementing the hybrid attack, combining the progressive BKZ with the Meet-In-the-Middle Attack. For an introduction to this sort of attack, see [12].

References

[1] Yoshinori Aono. A Faster Method for Computing Gama-Nguyen-Regev's Extreme Pruning Coefficients. *CoRR*, abs/1406.0342, 2014.

[2] Yoshinori Aono, Yuntao Wang, Takuya Hayashi, and Tsuyoshi Takagi. Improved Progressive BKZ Algorithms and Their Precise Cost Estimation by Sharp Simulator. In *Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 789–819. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[3] Yuanmi Chen. *Lattice Reduction and Concrete Security of Fully Homomorphic Encryption*. PhD thesis, Paris Diderot University, 2013.

[4] Yuanmi Chen and Phong Quang Nguyen. BKZ 2.0: Better Lattice Security Estimates. In *Advances in Cryptology – ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, pages 1–20. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[5] Darmstadt Technical University. TU Darmstadt SVP Challenge. `https://www.latticechallenge.org/svp-challenge/index.php`, 2016. Online; accessed: 2016-08-23.

[6] Nikesh Dattani and Nathaniel Bryans. Quantum Factorization of 56153 With Only 4 qubits. *CoRR*, abs/1411.6758, 2014.

[7] The Sage Developers. Sagemath, the Sage Mathematics Software System (Version 7.1). `http://www.sagemath.org`, 2016. Online; accessed: 2016-08-23.

[8] Nicolas Gama, Phong Quang Nguyen, and Oded Regev. Lattice Enumeration Using Extreme Pruning. In *Advances in Cryptology – EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 – June 3, 2010. Proceedings*, pages 257–278. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[9] Craig Gentry. Key Recovery and Message Attacks on NTRU-Composite. In *Advances in Cryptology — EUROCRYPT 2001: International Conference on*

the Theory and Application of Cryptographic Techniques Innsbruck, Austria, May 6–10, 2001 Proceedings, pages 182–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[10] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing Blockwise Lattice Algorithms Using Dynamical Systems. In *Advances in Cryptology – CRYPTO 2011: 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, pages 447–464. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[11] Jeffrey Hoffstein, Jill Pipher, and Joseph Hillel Silverman. NTRU: A Ring-based Public-key Cryptosystem. In *Algorithmic Number Theory: Third International Symposiun, ANTS-III Portland, Oregon, USA, June 21–25, 1998 Proceedings*, pages 267–288. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[12] Nick Howgrave-Graham. A Hybrid Lattice-Reduction and Meet-in-the-Middle Attack Against NTRU. In *Advances in Cryptology - CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings*, pages 150–169. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[13] Security Innovation. The NTRU Challenge. `https://www.securityinnovation.com/products/encryption-libraries/ntru-crypto/ntru-challenge`, 2016. Online; accessed: 2016-02-06.

[14] Security Innovation. NTRU PKCS Tutorial. `https://assets.securityinnovation.com/static/downloads/NTRU/resources/NTRU-PKCS-Tutorial.pdf`, 2016. Online; accessed: 2016-06-29.

[15] Security Innovation. The Solved NTRU Challenges. `https://assets.securityinnovation.com/static/downloads/NTRU/challenge/ntru-challenge-parameter-sets-and-public-keys-answers.pdf`, 2016. Online; accessed: 2016-08-04.

[16] Arjen Lenstra, Hendrik Lenstra, and László Lovász. Factoring Polynomials With Rational Coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[17] Mingjie Liu and Phong Quang Nguyen. Solving BDD by Enumeration: An Update. In *Topics in Cryptology – CT-RSA 2013: The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings*, pages 293–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[18] Joseph Hillel Silverman Nick Howgrace-Graham and William Whyte. A Meet-In-The-Middle Attack on an NTRU Private Key - NTRU Cryptosystems Technical Report # 004, Version 2. `https://www.securityinnovation.com/uploads/Crypto/NTRUTech004.pdf`, 2003. Online; accessed 2016-03-14.

[19] Ron Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[20] Claus Peter Schnorr. A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms. *Theor. Comput. Sci.*, 53(2-3):201–224, August 1987.

[21] Claus Peter Schnorr. Lattice Reduction by Random Sampling and Birthday Methods. In *STACS 2003: 20th Annual Symposium on Theoretical Aspects of Computer Science Berlin, Germany, February 27 – March 1, 2003 Proceedings*, pages 145–156. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[22] Claus Peter Schnorr and M. Euchner. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. *Math. Program.*, 66(2):181–199, September 1994.

[23] Cybersecurity Research Institute in National Institute of Information Security Fundamentals Laboratory and Communications Technology. Progressive BKZ Library, Version 1.1. `http://www2.nict.go.jp/security/pbkzcode/index.html`, 2016. Online; accessed 2016-06-29.

[24] Peter Williston Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *FOCS*, pages 124–134. IEEE Computer Society, 1994.

[25] Victor Shoup. NTL: A Library for doing Number Theory, Version 9.7.0. `http://www.shoup.net/ntl/download.html`, 2016. Online; accessed 2016-06-06.

[26] Joseph Hillel Silverman. Almost Inverses and Fast NTRU Key Creation - NTRU Cryptosystems Technical Report # 014, Version 1. `https://www.securityinnovation.com/uploads/Crypto/NTRUTech009.pdf`, 1999. Online; accessed 2016-03-09.

[27] The FPLLL Development Team. FPLLL, a Lattice Reduction Library. `https://github.com/fplll/fplll`, 2016. Online; accessed: 2016-08-23.

[28] Joop van de Pol. Lattice-based Cryptography. Master's thesis, Eindhoven University of Technology, 2011.

[29] Wikipedia. Advanced Encryption Standard. `https://en.wikipedia.org/wiki/Advanced_Encryption_Standard`, 2016. Online; accessed 2016-02-29.

[30] Wikipedia. NTRUEncrypt. `https://en.wikipedia.org/wiki/NTRUEncrypt`, 2016. Online; accessed 2016-03-01.

Appendix A

Toy Example of Reduction of an NTRU Lattice

To give the reader a feel for how lattice reduction works, a couple of different reductions on a small NTRU example are shown. This can be used to help to understand Chapter 3 and 4. The example is from [14]. Bob uses the system parameters $(n, p, q) = (11, 3, 32)$ and the private key $f(x) = -1 + X + X^2 - X^4 + X^6 + X^9 - X^{10}$ and $g(x) = -1 + X^2 + X^3 + X^5 - X^8 - X^{10}$. First he calculates

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

Then the public key is

$$h(x) = 3 \cdot f_q(x) * g(x) \pmod{32} = 8 + 25X + 22X^2 + 20X^3 + 12X^4 + \\ + 24X^5 + 15X^6 + 19X^7 + 12X^8 + 19X^9 + 16X^{10}.$$

Multiplying this polynomial by $p^{-1} \pmod{q} = 3^{-1} \pmod{32} = 11$ we get

$$h'(x) = 11 \cdot h(x) \pmod{32} = 24 + 19x + 18x^2 + 28x^3 + 4x^4 + 8x^5 + 5x^6 + \\ + 17x^7 + 4x^8 + 17x^9 + 16x^{10},$$

which is used to define the following NTRU lattice basis

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 24 & 19 & 18 & 28 & 4 & 8 & 5 & 17 & 4 & 17 & 16 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 16 & 24 & 19 & 18 & 28 & 4 & 8 & 5 & 17 & 4 & 17 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 17 & 16 & 24 & 19 & 18 & 28 & 4 & 8 & 5 & 17 & 4 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 17 & 16 & 24 & 19 & 18 & 28 & 4 & 8 & 5 & 17 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 17 & 4 & 17 & 16 & 24 & 19 & 18 & 28 & 4 & 8 & 5 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 5 & 17 & 4 & 17 & 16 & 24 & 19 & 18 & 28 & 4 & 8 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 8 & 5 & 17 & 4 & 17 & 16 & 24 & 19 & 18 & 28 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 4 & 8 & 5 & 17 & 4 & 17 & 16 & 24 & 19 & 18 & 28 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 28 & 4 & 8 & 5 & 17 & 4 & 17 & 16 & 24 & 19 & 18 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 18 & 28 & 4 & 8 & 5 & 17 & 4 & 17 & 16 & 24 & 19 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 19 & 18 & 28 & 4 & 8 & 5 & 17 & 4 & 17 & 16 & 24 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 32
\end{pmatrix}.$$

On vector form the private key is
$(f, g) = [-1, 1, 1, 0, -1, 0, 1, 0, 0, 1, -1, -1, 0, 1, 1, 0, 1, 0, 0, -1, 0, -1]$. After an LLL-reduction of this basis, made in NTL[25], with default parameters, we get

$$\begin{pmatrix}
-1 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 1 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & -1 & -1 & 1 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & 1 & 0 \\
-1 & 1 & 1 & -1 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 1 \\
0 & -1 & 0 & 1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 1 \\
-1 & 0 & 1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 1 & 1 \\
-1 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 1 & 1 & -1 & -1 & -1 & 0 & -1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
-1 & -1 & 1 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & -1 & 0 \\
1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & -1 & -1 & 1 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -1 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 1 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 1 & 0 & 1 \\
0 & -1 & 0 & -2 & -1 & 1 & -3 & 3 & 3 & 2 & -2 & 2 & -4 & 4 & -3 & -4 & 4 & 2 & 2 & 0 & -7 & 4 \\
3 & -3 & -1 & -2 & -5 & 3 & 0 & 6 & -2 & 5 & -1 & 0 & 0 & 2 & -2 & -1 & 3 & -5 & -1 & 5 & -2 & 1 \\
0 & -1 & -1 & 1 & -2 & 2 & 2 & 3 & -1 & 0 & -2 & 5 & -3 & -4 & 3 & 2 & 1 & -1 & -7 & 5 & 3 & -4 \\
4 & 2 & 3 & -2 & 1 & 1 & -1 & -1 & 0 & 0 & -1 & 2 & 0 & -7 & 5 & 3 & -3 & 4 & -3 & -5 & 3 & 1 \\
1 & -1 & 3 & -3 & -3 & -2 & 2 & 0 & 1 & 0 & 2 & 4 & -4 & -2 & -2 & 0 & 7 & -4 & -2 & 4 & -4 & 3 \\
3 & -7 & 3 & -4 & 1 & -2 & 4 & 4 & -2 & 1 & -4 & 0 & -1 & -2 & -1 & 1 & 4 & -3 & -3 & 0 & 0 & 5 \\
3 & -2 & 6 & -2 & 5 & 1 & 2 & -5 & -1 & -2 & -3 & 3 & -4 & -2 & 5 & -3 & 1 & 0 & -1 & 2 & -2 & 1 \\
-5 & 2 & 0 & 7 & -2 & 5 & 0 & 2 & -4 & 0 & -1 & 0 & 3 & -4 & -1 & 5 & -3 & 1 & -1 & -1 & 2 & -1 \\
1 & -5 & 3 & -6 & 1 & -1 & 4 & 2 & 2 & 4 & -1 & 4 & 2 & -6 & 2 & -2 & 0 & 1 & -2 & 3 & 0 & -2 \\
6 & -2 & 6 & 0 & 1 & -4 & 0 & -2 & -4 & 3 & -1 & -2 & 4 & -3 & 0 & -1 & -1 & 3 & -1 & 1 & 4 & -4 \\
-2 & 2 & -1 & 3 & 1 & 1 & -1 & -4 & 1 & -2 & 3 & -6 & -1 & -2 & -5 & -3 & -6 & -1 & 1 & -6 & -1 & -2
\end{pmatrix}.$$

Next we do a BKZ-8 reduction of the basis in NTL with default parameters and end up with the final basis

$$\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 1 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & -1 & -1 & 1 & 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 & 1 & 0 \\
-1 & 1 & 1 & -1 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 1 \\
0 & -1 & 0 & 1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & 0 & -1 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 1 & 1 & 0 \\
-1 & -1 & 1 & 1 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & -1 & 0 \\
-1 & 0 & 1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & 1 & 1 \\
0 & 0 & -2 & 1 & 0 & -2 & 0 & -1 & 0 & 1 & -2 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & -1 & -1 & 0 & 1 & 0 & -1 & 0 & 0 & -1 & 1 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & 1 & 0 & 1 \\
0 & -1 & 1 & 1 & -1 & -1 & 0 & 1 & 0 & -1 & 0 & 1 & 0 & 1 & 1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 \\
0 & -1 & 0 & -2 & -1 & 1 & -3 & 3 & 3 & 2 & -2 & 2 & -4 & 4 & -3 & -4 & 4 & 2 & 2 & 0 & -7 & 4 \\
-5 & 2 & 0 & 7 & -2 & 5 & 0 & 2 & -4 & 0 & -1 & 0 & 3 & -4 & -1 & 5 & -3 & 1 & -1 & -1 & 2 & -1 \\
5 & -3 & 6 & -1 & 1 & -4 & -2 & -2 & -4 & 1 & -1 & -2 & 6 & -2 & 2 & 0 & -1 & 2 & -3 & 0 & 2 & -4 \\
-3 & -2 & 2 & 0 & 1 & 0 & 2 & 1 & -1 & 3 & -3 & 0 & 7 & -4 & -2 & 4 & -4 & 3 & 4 & -4 & -2 & -2 \\
-2 & 3 & -3 & 2 & -2 & 3 & 0 & 3 & 2 & 0 & -2 & 0 & -5 & -2 & -3 & -6 & -1 & -1 & -5 & -1 & -7 & -1 \\
1 & 2 & -3 & -1 & -2 & 3 & 0 & 0 & 2 & 2 & 1 & -3 & -1 & -2 & 0 & 7 & -5 & -3 & 3 & -4 & 3 & 5 \\
0 & -7 & 2 & -5 & 0 & -2 & 4 & 0 & 1 & 5 & -2 & 4 & 1 & -5 & 3 & -1 & 1 & 1 & -2 & 1 & 0 & -3 \\
0 & -1 & -5 & 2 & 0 & 7 & -2 & 5 & 0 & 2 & -4 & 2 & -1 & 0 & 3 & -4 & -1 & 5 & -3 & 1 & -1 & -1 \\
3 & -3 & 0 & -1 & -2 & -1 & 0 & 0 & -4 & 2 & 3 & -6 & 5 & 3 & -3 & 3 & -3 & -5 & 4 & 0 & 2 & 0 \\
0 & -2 & 4 & 0 & 1 & 5 & -2 & 0 & -7 & 2 & -5 & -1 & 1 & 1 & -2 & 1 & 0 & -3 & 4 & 1 & -5 & 3 \\
3 & 0 & 0 & 2 & 2 & 1 & 1 & 2 & -3 & -1 & -2 & -5 & -3 & 3 & -4 & 3 & 5 & -3 & -1 & -2 & 0 & 7
\end{pmatrix},$$

since doing a BKZ-reduction with a bigger blocksize does not change this particular basis. Also notice that the first vector is the shortest, but can not be the private key. The problem of searching for the private key in the lattice is not the same, as the problem of finding the shortest vector in the lattice. The problems are very similar but not identical. None of the following vectors are equal to the private key either.

Even after having made an excellent reduction on the basis, it is still not entirely obvious how to find the private key in general. It is however not the focus of the thesis to do that extra step and will not be further discussed.

Just to show that the private key indeed is a lattice vector, we use the fact that we know the private key and notice that the private key equals the tenth row but with opposite signs.

Appendix B

List of Acronyms

BDD Bounded Distance Decoding 40

BKZ Block Korkine Zolotarev 20

CVP Closest Vector Problem 15

FEC Full Enumeration Cost 31

GH Gaussian Heuristic 15

GSA Geometric Series Assumption 32

GSO Gram-Schmidt Orthogonalization 19

LLL Lenstra-Lenstra-Lovász 19

NTL Number Theory Library 27

NTRU Nth Degree Truncated Polynomial 9

RSA Rivest-Shamir-Adleman 4

SVP Shortest Vector Problem 15

Appendix C

Populärvetenskaplig sammanfattning

Gitterbaserade attacker mot potentiellt kvantsäkert kryptosystem

För att kunna bedömma om ett nytt kryptosystem är säkert och i så fall hur stor krypteringsnyckel systemet behöver för att vara säkert måste attacker mot systemet analyseras ingående.

Att kunna skicka krypterade meddelanden är inte längre bara angeläget för militärer och underrättstjänster. Varje gång du betalar räkningar via din internetbank eller handlar varor över internet vill du kryptera datan du sänder så att inte obehöriga kan ta del av känslig information.

Symmetrisk kryptering

Tänk dig situationen att Alice vill skicka ett hemligt meddelande till Bob. Inom det som kallas symmetrisk kryptering använder Alice då ett hemligt ord (nyckel) för att göra meddelandet oläsligt (kryptera). Bob använder sen samma nyckel för att göra det krypterade meddelandet läsligt igen (dekryptera). Med symmetrisk kryptering kan stora mängder data skickas snabbt och säkert. Ett problem är dock hur Alice och Bob ska komma överens om en gemensam nyckel. Ett sätt detta kan göras på är med asymetrisk kryptering.

Asymetrisk kryptering

Inom asymetrisk kryptering har Bob två nycklar, en publik nyckel som vem som helst kan se och en privat nyckel som bara Bob kan se. Alice skickar nu ett krypterat meddelande med hjälp av Bobs publika nyckel. Bob dekrypterar sen meddelandet med hjälp av sin privata nyckel.

Kvantdatorer

För att kunna göra det krypterade meddelandet läsligt igen utan tillgång till den privata nyckeln måste en attackerare lösa ett matematiskt problem. Det mest använda kryptosystemet är RSA och det matematiska problemet för att knäcka RSA är att faktorisera en produkt av två stora primtal.

En kvantdator är en dator baserad på kvantvbitar, som kan anta värden mellan 0 och 1, till skillnad från bitarna i en vanlig dator som bara kan ha värdena 0 eller 1. Idag kan kvantdatorerna bara hantera ett fåtal kvantbitar. Men får vi storskaliga kvantdatorer i framtiden visar det sig att det är lätt att faktorisera stora tal och då behövs en ersättare till RSA.

NTRU

Ett gitter är en matematisk struktur med diskret utspridda punkter. I tre dimensioner är punkterna i ett gitter fördelade som atomerna i en kristall. I högre dimensioner, givet en målpunkt någonstans i ett gitter, visar det sig vara ett svårt matematiskt problem att hitta den punkt i gittret som är närmast målpunkten. NTRU är ett kryptosystem baserat på detta problem, med gitter på en speciell form. Ju längre den publika nyckeln är desto högre är dimensionen på detta gitter och desto svårare är NTRU att knäcka. Å andra sidan tar en onödigt lång nyckel onödigt mycket plats och gör att kryptering och dekryptering tar onödigt mycket tid.

NTRU-utmaningarna

I motsats till RSA känner man i nuläget inte till något sätt att snabbt knäcka NTRU med hjälp av kvantdatorer. Men NTRU är inte lika väl studerat som RSA. För at försäkra sig om att det verkligen inte finns någon snabb lösning och för att bättre kunna bedömma hur lång nyckel systemet behöver utlyste företaget bakom NTRU de så kallade NTRU-utmaningarna. Tjugosex olika publika nycklar av ökande längd publicerades och en liten ekonomisk ersättning utdelas till den första personen som räknar ut den privata nyckeln tillhörande respektive publik nyckel.

Resultat

Med hjälp av ett nyutvecklat mjukvarubibliotek anpassat för att leta efter korta vektorer i ett gitter försökte jag lösa så många NTRU-utmaningar som möjligt. Jag lyckades lösa de första 5 utmaningarna. Med mer tid och/eller datorkraft hade jag sannolikt även kunnat lösa utmaning 6. Utmaning 7 är den svåraste utmaningen någon löst överhuvudtaget. Det ska sägas att utmaningarna ökar snabbt i svårighetsgrad och att resultatet från det här projektet inte ändrar gränsen för vilken nyckellängd som är tillräckligt säker.